Personal Computer
PC-8801mkII
NEC
プログラミング入門
ナツメ社編集部＝編著
ナツメ社

Personal Computer

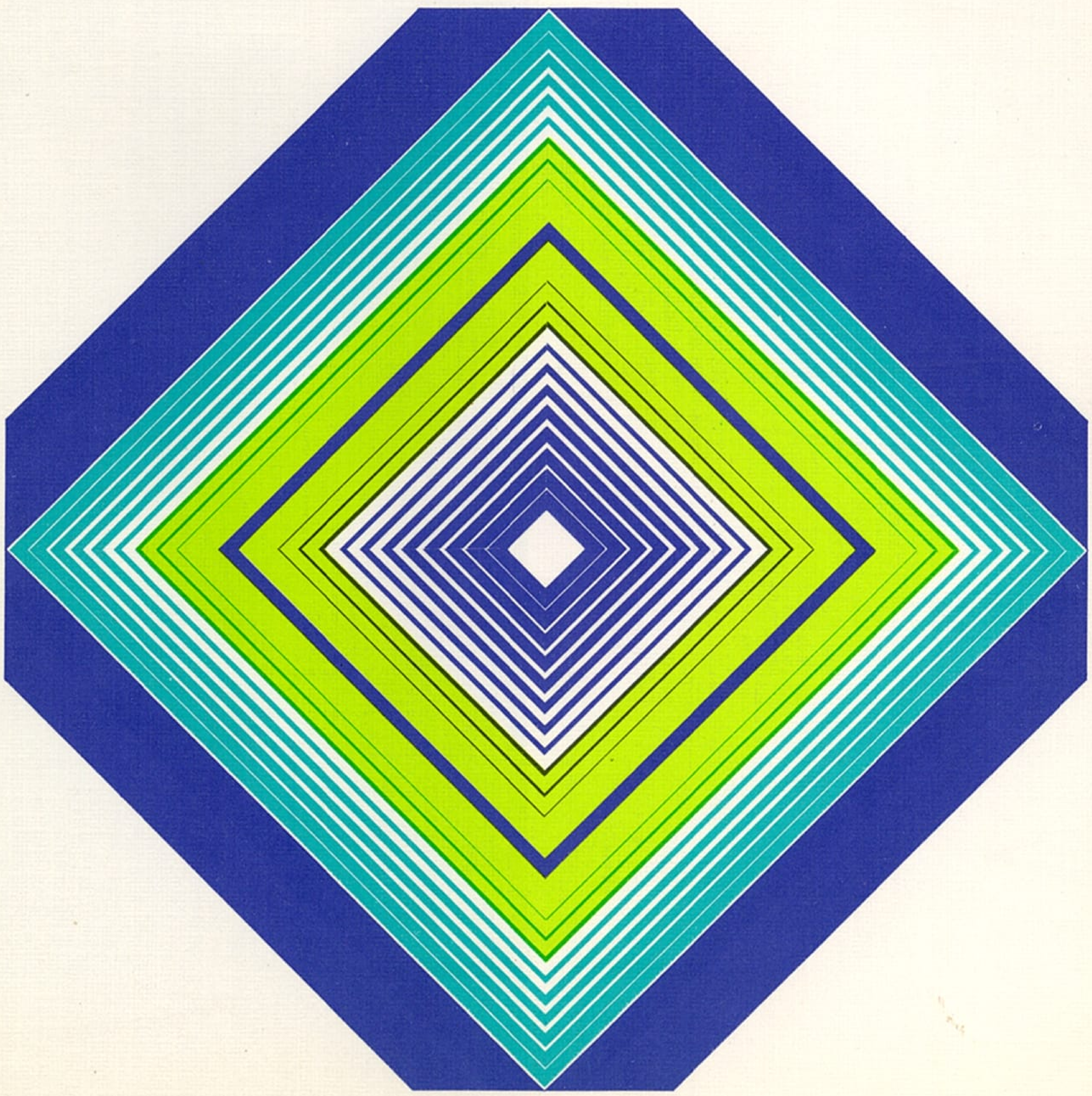
PC-8801 mkII

NEC

プログラミング入門

ナツメ社編集部＝編著

ナツメ社



[本書を読まれる方へ]

⊙この本はPC-8801mkIIの
操作法およびプログラミングの手法を解説したものです。
キー操作のABCから実用的なプログラムまで、
実行画面をできるだけ多く載せながら解説しましたので、
コンピュータとは何かをまったく知らない人でも
抵抗なく読み進んでいくことができます。
⊙英単語をいくつ丸暗記しても
英会話を話せることにはならないように、
コマンド、ステートメントを覚えただけでは、
プログラム中での使い方がわからなければ、
まったく実用になりません。
⊙この本は、こういった問題を解決し、
みなさんができるだけニーズに合った
プログラム作りができるように、プログラムを
白紙の状態から作りつつ執筆されていますので、
プログラム完成までの紆余曲折を
はっきりつかんでいただけると思います。
⊙そして、こういった試行錯誤こそが、
プログラミングには一番大切なことなのです。
みなさんのコンピュータ購入の動機はまちまちでも、
プログラミングマスターまでは
同じ道程をたどらなければなりません。
⊙さあ、みなさんそろって、がんばってみませんか?
最後には、タートルグラフィック、
サウンド機能を生かした楽しいゲームが待っています。

cover design——nobuo nakagaki＋yoshifumi hayase

Personal Computer

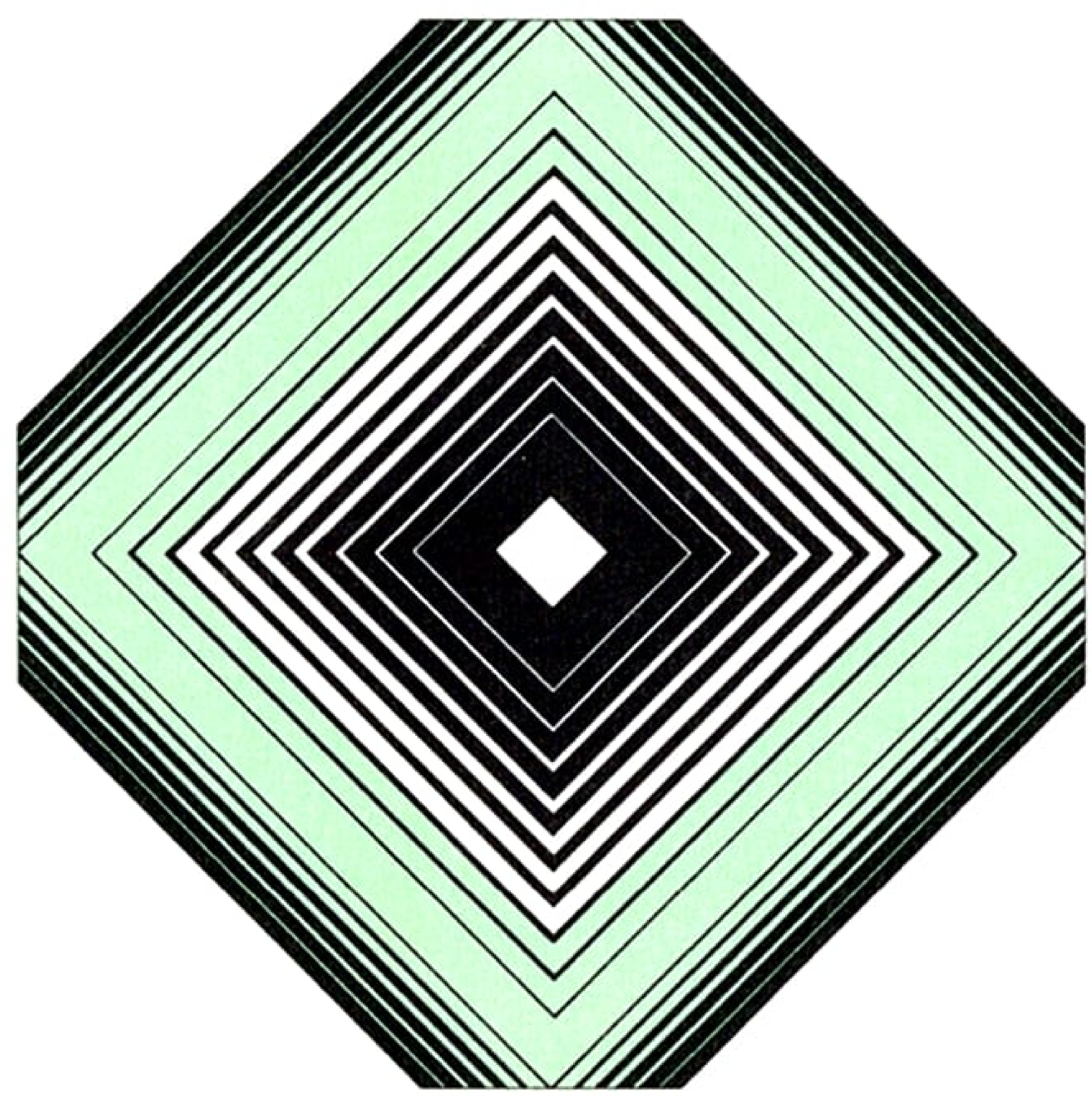
PC-8801 mkII

NEC

プログラミング入門

ナツメ社編集部＝編著

ナツメ社



この本を読む前に

この本は、タイトルにあるようにＮＥＣのパソコン

PC-8801mkII

の操作法およびプログラミングの手法を解説したものです。

入門書の形をとっていますので、キー操作など、ごく基本的な事項から説明していますが、すでにこの辺をマスターしている方は、第３章からお読みください。

この本でサンプルとして取り上げたテーマは、ビジネス、ゲーム、教育で、それぞれに１章ずつさいて解説しました。分野は違っても、それぞれに共通する有効なコマンド、ステートメントがありますので、ぜひ、どの章も実践されることをおすすめします。

英単語をいくつ丸暗記しても英会話を話せることにはならないと同じように、コマンド、ステートメントを覚えても、プログラム中での使い方がわからなければ実用になりません。

また、１から100まで加算するプログラムとか、長方形を描くプログラムなど、こまぎれのプログラムをマスターしても、実用には結びつきにくいようです。

この本は、こういった問題を解決し、みなさんができるだけ実用的なプログラム作りができるように、プログラムを白紙の状態から作りつつ執筆しましたので、プログラム完成までの紆余曲折をはっきりつかんでいただけると思います。

プログラミングの入門書を見ると、整然としたプログラムリストが並んでいますが、最初からこんなにうまくはいきません。この本でぜひプログラミングのプロセスを把握していただき、仕事とか趣味とかに生かしていただければと思います。

なお本書は、「ＰＣ-8801mkIIモデル30」をもとに執筆されていますが、必要に応じて「モデル10」「モデル20」についてもふれていますので、そのままご利用になれます。

ナツメ社編集部

編集部より読者の方へ

この本は、前ページにも述べたように、私たちの身近なテーマをもとにプログラミングの実際を説明していますので、コマンド、ステートメントだけを並べた他の類書に比較して、きわめてわかりやすくなっています。途中で挫折してしまう人が多いようですが、本書の読者は、ぜひみなさんそろって終了し、次のステップへ進まれることを期待します。

本書を終えて次のステップへ進みたい方のために、ちょっとはやいのですが参考書を紹介しておきます。

❶ ゲームを目ざしたい方は

ＰＣ－8001　はるみのゲームライブラリー
ＰＣ－8001　はるみのゲームライブラリー（ＰＡＲＴ２）
ＰＣ－8001　はるみのプログラミングレッスン

これらはＰＣ－8001用ですがＰＣ－8801マークIIでも「NEW　ON 1　↵」を実行することによりゲームが楽しめます。

❷ データベースを勉強したい方は

ＰＣ－8801mkIIデータファイル入門（Ｉ）
ＰＣ－8801mkIIデータファイル入門（II）

❸ ビジネスを志したい方は

ＰＣ－8801　ビジネスプログラミング

❹ マシン語を勉強したい方は

ＰＣ－8801mkII　マシン語活用入門
ＰＣ－8801　アセンブリ言語プログラミング入門

❹ グラフィックスの世界を楽しみたい方は

ＰＣ－8801mkII　グラフィックス入門

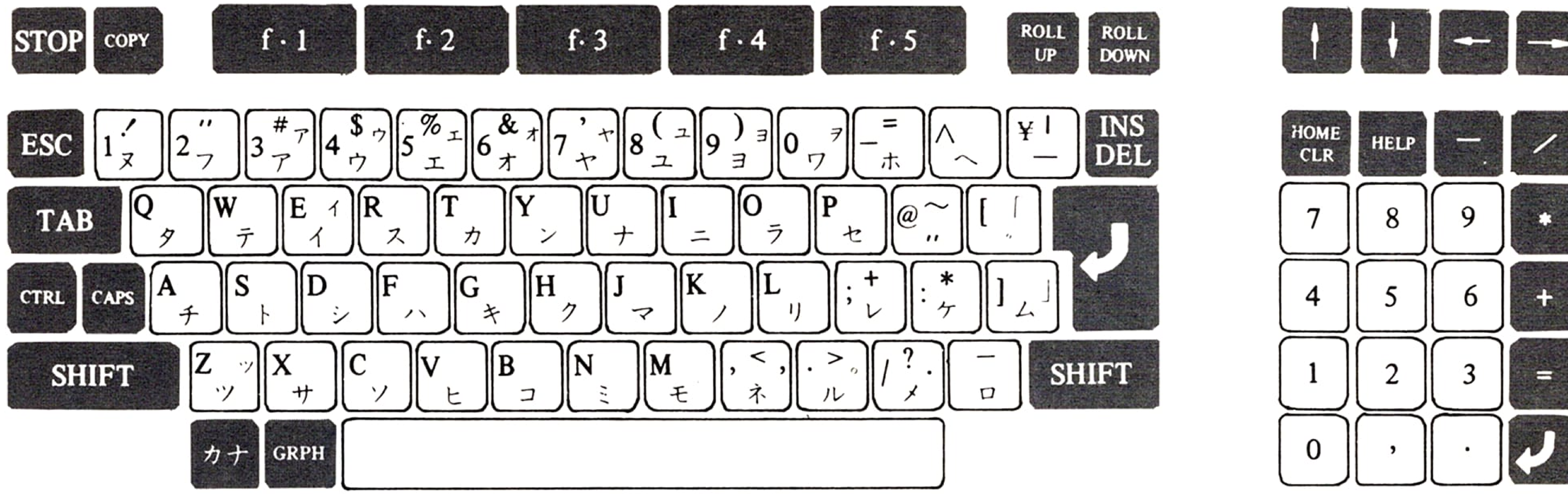
❺ ワードプロセッサの考え方を知りたい方は

ＰＣ－8801　漢字ワードプロセッサ入門

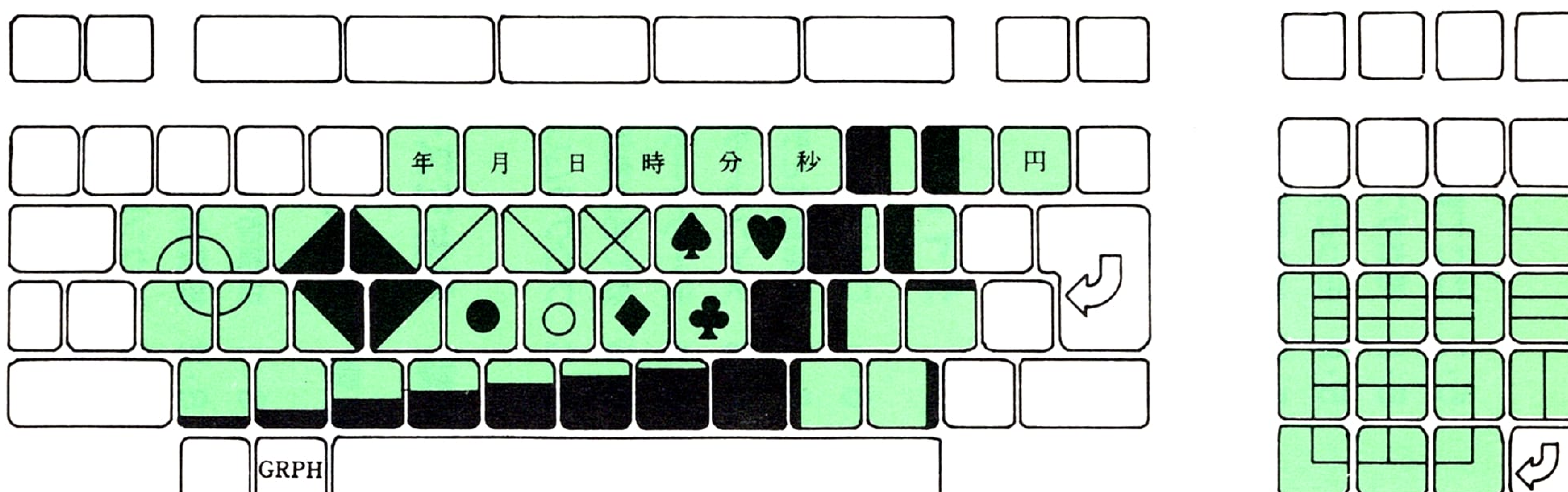
以上、いずれもナツメ社発行です。

編集部

もくじ



●キーボードのキー配列●



●グラフィックシンボルのキー配列●

第1章

とにかく
スイッチON

1·1 電源を入れてみよう

電源投入の順序

パソコン本体とキーボード、ディスプレイとの接続はできていますか？　まだの人は、ユーザーズマニュアルを見て接続してください。また、カセットテープレコーダ、プリンタはまだ使いませんから、電源は入れなくてもけっこうです。

準備ができましたら、次の順序でスイッチをＯＮにします。

❶ディスプレイのスイッチＯＮ

これを一番最初に行うのは、あたたまるまでに少し時間がかかるからです。１分間くらい待ってから次のステップへ進みましょう。

❷システムディスクをセット

システムディスク
ドライブ１

ディスク内蔵タイプの本体をご使用の方は、システムディスクを、本体前面の「DRIVE 1」という文字が書いてある側（これをドライブ１といいます）の挿入口から、ラベルを上にして差し込んでください。突き当たるまで入れましたら、入口のストッパーをひねって固定してください。

システムディスクというのはおわかりですね。標準添付の２つ折りジャケットの中に、２枚の四角い磁気盤が入っています。一方のラベルに「N88DISK−BASIC　SYSTEM　DISK」と書いてありますね。これがシステムディスクです。これは、ディスクを起動させるためのデータが書き込まれています。

もう一方のラベルには、「GAME」と書いてあります。このディスクにはゲームプログラムがおさめられています。これはあとで楽しむことにしましょう。

❸本体のスイッチＯＮ

初期画面（モデル10の場合）

本体の電源を入れると、ディスク内蔵タイプでない場合、ディスプレイ上に次のような英文のメッセージが表示されます。

```
How many files(0-15)? ■
```

メッセージの一番うしろで、何やら四角いもの*がピコピコ点滅して

＊これをカーソルといいます。

います。そしてその前にクエッションマークがありますね。英語ではクエッションマークがつくと疑問形式ですね。つまりこれは、あなたにコンピュータが問いかけているのです。

この意味は、「いくつのファイルを使いますか」ということですが、いまは理解できなくてもかまいません。とりあえず、キーボード中央の右側、ＳＨＩＦＴと書いてあるキー（これをシフトキーといいます）の上の矢印のついた大きなキーを押してください。

この、道路表示のような矢印のキーはリターンキーと呼ばれ、一番大切なキーです。

リターンキー

本書では以後、リターンキーは のマークを使うことにします。

さあ、リターンキーを押しましたか？

押すと同時に、画面は次のようになります。

```
How many files(0-15)?
NEC N-88 BASIC Version 1.3
Copyright (C) 1981 by Microsoft
56799 Bytes free
Ok
■

    load "    auto    go to    list    run
```

これで準備ができました。

初期画面（ディスク内蔵タイプの場合）

次は、ディスク内蔵のタイプの場合です。

本体の電源を入れますと、ディスクが起動する音がして、しばらくしてから次のようなメッセージが表示されます。

```
Disk version [Aug 19,1983]
How many files(0-15)? ■
```

How many files(0-15)?

カーソル

という表示のうしろで四角いもの（カーソルといいます）が点滅しています。あなたに「ファイルは同時にいくつ使いますか」と問いかけているのですね。ファイルといわれてもいまはピンとこないかもしれません。とりあえず、キーボードの右側に数字のキーのグループがありますから、この中から 5 というキーを見つけて押してください。

リターンキー

●リターンキーはもう1つ右下すみにもあります。どちらも同じです。便利なほうを使ってください。

次に、キーボード中央の右側、ＳＨＩＦＴと書いてあるキーの上の矢印のついた大きなキーを押してください（この、道路標示のような矢印のキーはリターンキーと呼ばれ、一番大切なキーです。本書ではリターンキーを押すことを ⏎ のマークを使って表します）。

画面は次のようになりました。

```
Disk version [Aug 19,1983]
How many files(0-15)? 5
NEC N-88 BASIC Version 1.3
Copyright (C) 1981 by Microsoft
44581 Bytes free
Ok
■

 load "   auto   go to   list   run
```

メッセージの一番下に「Ok」と表示がでて、その下でカーソルが点滅しています。「Ok」ということ、つまりコンピュータが「あなたからのいろいろな命令（コマンド）を受け入れる準備ができました。オーケーですよ」といっているのです。

1・2　キー操作を覚えよう

オーケーといわれても、これから何をしたらいいのか、私たちはとまどってしまいますね。しかし、ただ考えていても始まりません。キーボードのキーを押して何かやってみましょう。

カーソルを動かす

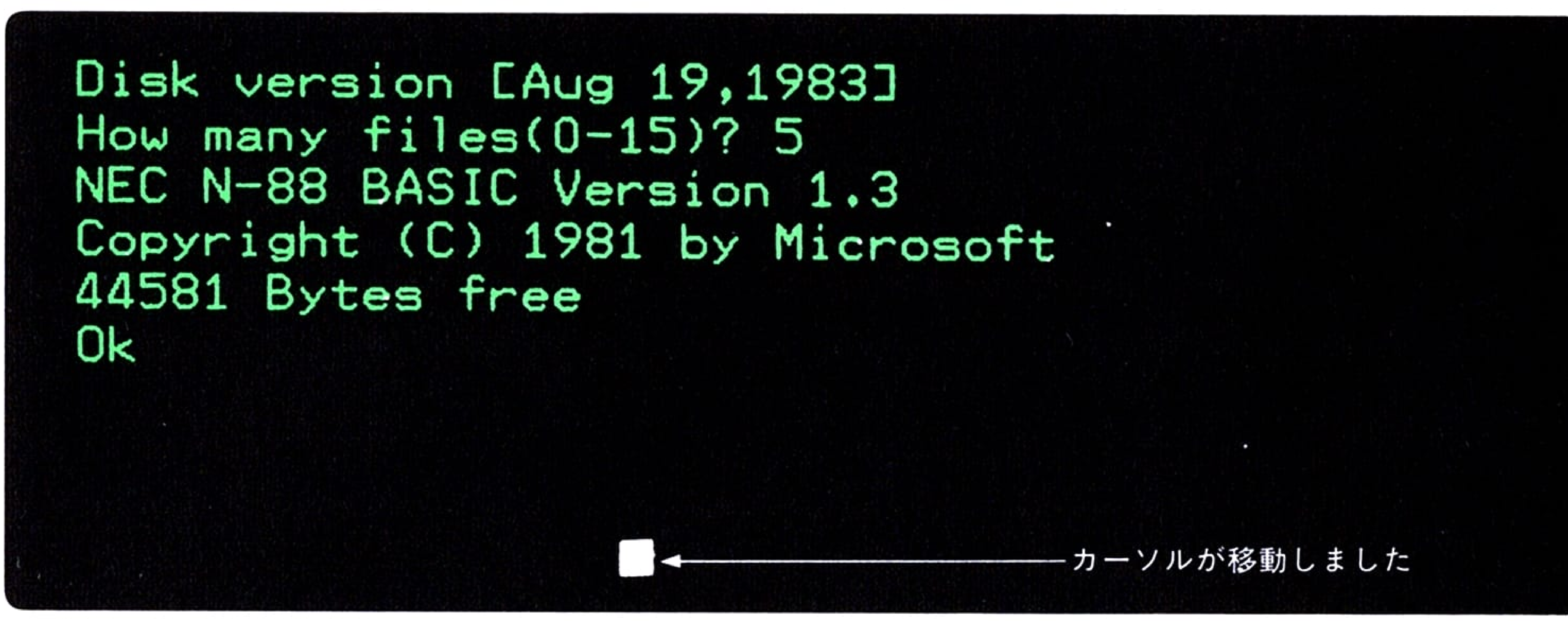
カーソル

先ほど説明したように、ディスプレイ上で点滅している四角をカーソルといいます。ここに、これから先へ進むヒントがありそうです。

キーボードの右上すみに

↑　↓　←　→

という4つのキーがあります。このキーを、どれでもいいですから何回か押してみてください。

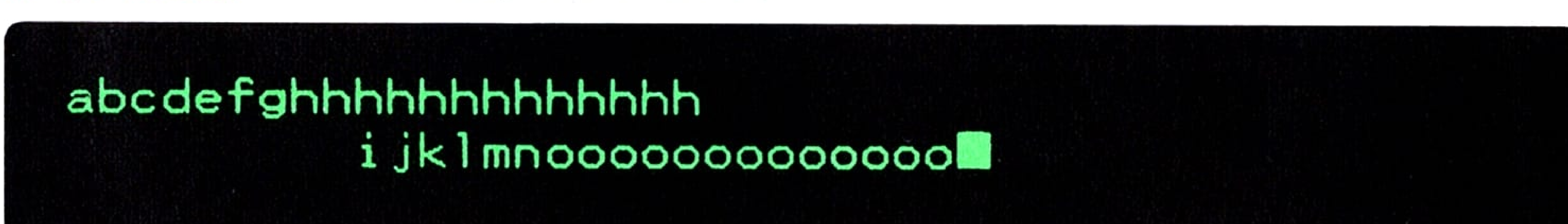


カーソル移動キー

どうですか？　カーソルがキーに示されている矢印の方向に動きましたね。この4つのキーをカーソル移動キーといい、これもよく使う大切なキーです。

リピート機能

さあ、それでは数字とかアルファベットの書いてあるキーを適当に押してみてください。カーソルのある位置に押したキーの文字が表示されて、カーソルはその隣りに移動するのがわかるでしょう。キーを押し続けるとその間、同じ文字を表示し続けます。これをリピート機能（くり返し）といい、大変便利なものです。



画面をキレイにする

キーボードを見ると、ずいぶんいろいろなキーが並んでいますが、ここでは、よく使うキーについてだけ説明することにします。

テンキー

ホームクリアキー

キーボード右側に数字の書かれたキーが集まった四角の部分（テンキーといいます）がありますが、その上のほうに「ＨＯＭＥ」と「ＣＬＲ」の２つの語が書いてあるキーがあるでしょう。これは「ホームクリアキー」と呼ばれ、これを押すと画面がクリア、つまり表示されている文字がキレイに消されてしまいます。

ホームポジション

いま、あなたのディスプレイにはいろいろな文字が表示されていると思います。クリアキーを押してみてください。画面の文字が消えてカーソルが左上すみに移動して点滅しているでしょう。この左上すみを「ホームポジション」といいます。

このようにホームクリアキーは画面の表示を消去して、カーソルをホームポジションに移す機能をもっています。

また、何かのつごうで画面の表示は残しておきたいときは、キーボード下の左右にありますＳＨＩＦＴと書かれた (SHIFT) キーを押しながらホームクリアキーを押せば、カーソルだけがホームポジションに移動します。

数字を画面に表示する

数字を表示するには２つの方法があります。

キーボードをよくごらんください。Ⅷページの図〈キーの配列〉で白いキーの部分が２か所あります。

テンキー

右側のほうはテンキーといって、数字入力専用に使われます。パソコンによってはテンキーのない機種もありますが、ビジネスに使おう

右図について
- このキーだけを押すと❶が
- (SHIFT) キーを押しながらこのキーを押すと❷が
- (カナ) キーをロックしてこのキーを押すと❸が
- (カナ) キーをロックした状態で (SHIFT) キーを押しながらこのキーを押すと❹が

表示されます。

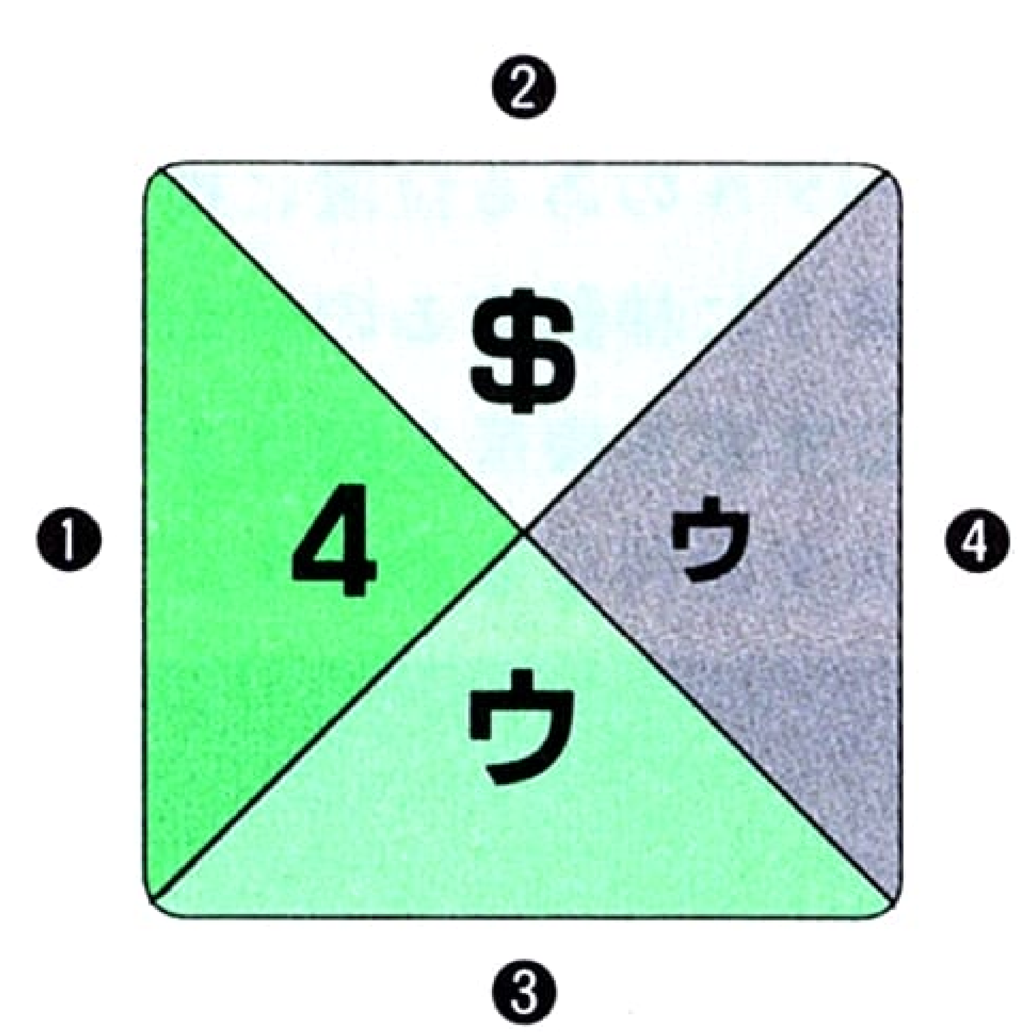



図 1


シャープ
ダブルクォーテーション
エクスクラメーション
ストップキー
エスケープキー
タブキー
コントロールキー
キャピタルロックキー
ドルマーク
パーセント
アンドマーク
アポストロフィ
マイナス
イコール
ベキ実記号
円マーク
音びき
濁点
半濁音符
マイナス
シフトキー
カナキー
グラフィックシンボルキー
スペースキー
コンマ
読点
小数点
句点
割り算記号
ナカグロ
シフトキー
リターンキー
コンマ
小数点
リターンキー
イコール
プラス記号
かけ算記号
わり算記号

とすると、このテンキーがないと大変不便に感じます。

それからもう１つ、左側の白い部分の一番上に数字が書かれたキーが横に並んでいます。このキーを押すと、書かれている数字が画面に表示されます。

どちらでも、使いやすいほうを使ってください。

アルファベットを表示する

アルファベットのキーはもうわかりますね。それぞれのキーには大文字が書き込まれています。しかし、このキーを押した場合、表示されるのは小文字です。

ちょっと押してみましょう。

```
abcdefghijklmn■
```

それでは SHIFT キーを押しながらアルファベットキーを押してみてください。大文字が表示されましたね。

大文字で画面にだしたい場合には、もう１つ方法があります。キーボード左側の下のほうにある CAPS キー（キャピタルロックキー）を押してから、アルファベットキーを押せばよいのです。

キャピタルロックキー

今度は、 CAPS キーをロックしたままの状態で、 SHIFT キーを押しながらアルファベットキーを押してみてください。小文字が表示されました。

ふたたびノーマルな状態にもどしたければ、もう一度 CAPS キーを押してロックをといてからアルファベットキーを押します。 SHIFT キーを使わずに小文字が表示されましたね。

あなたの名前などをローマ字で入力してみてください。

```
abcdefghijklmnABCDEF
ABCDEFGabcd
Natsumesha■
```

カタカナの表示

カタカナを画面にだしたいときは、キーボードの左下に カナ キーというのがありますから、これを押して（ロックして）、カタカナが書き込んであるキーを押します。 カナ キーは一度押せばロック

カナキー

されますから手をはなしてもだいじょうぶですよ。

6ページの図1をごらんください。

このキーの場合には❸の「ウ」が表示されます。

❹の「ゥ」のように少し小さい文字が必要なときはどうしたらよいでしょうか？　カナ キーをロックしたままの状態で、SHIFT キーを押しながら図1のキーを押します。

やってみましょう。どうですか？　うまくいきましたか？

カタカナ表示が不要になったら、カナ キーをもう一度押してロックをといておきましょう。

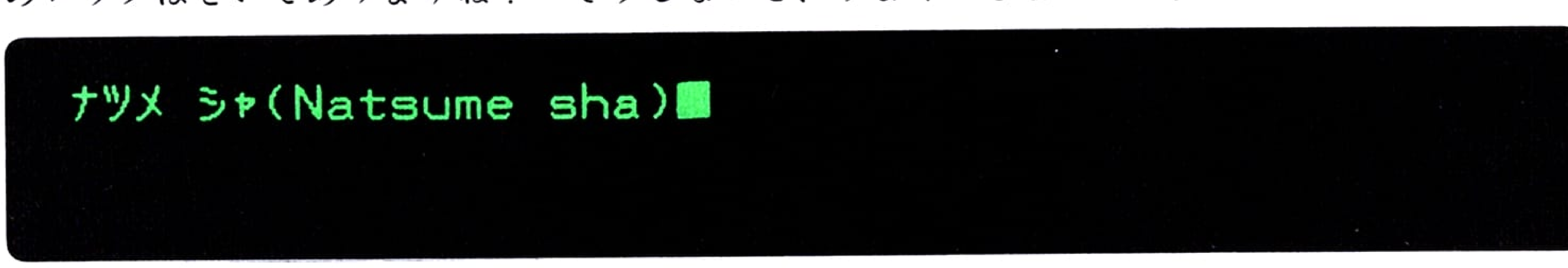
また、「バ」とか「パ」のような濁音、半濁音を表示したい場合には、まず カナ キーをロックして「ハ」をキーインし、次に大きいほうのリターンキーの左側にある濁点、半濁音符が書かれているキーを押します。つまり，濁音、半濁音を表示するには2字分が必要になります。ためしてみてください。

記号を表示したいとき

テンキー側にある「＝」とか「＋」の記号は、そのキーを押せば書かれている記号がそのまま表示されます。

6ページの図1のように、ドルマーク「＄」が必要な場合には、SHIFT キーを押しながらこのキーを押します。

ためしてみてください。カタカナ表示のときに押した カナ キーのロックはといてありますね？　そうしないと、うまくいきませんよ。



ファンクションキー

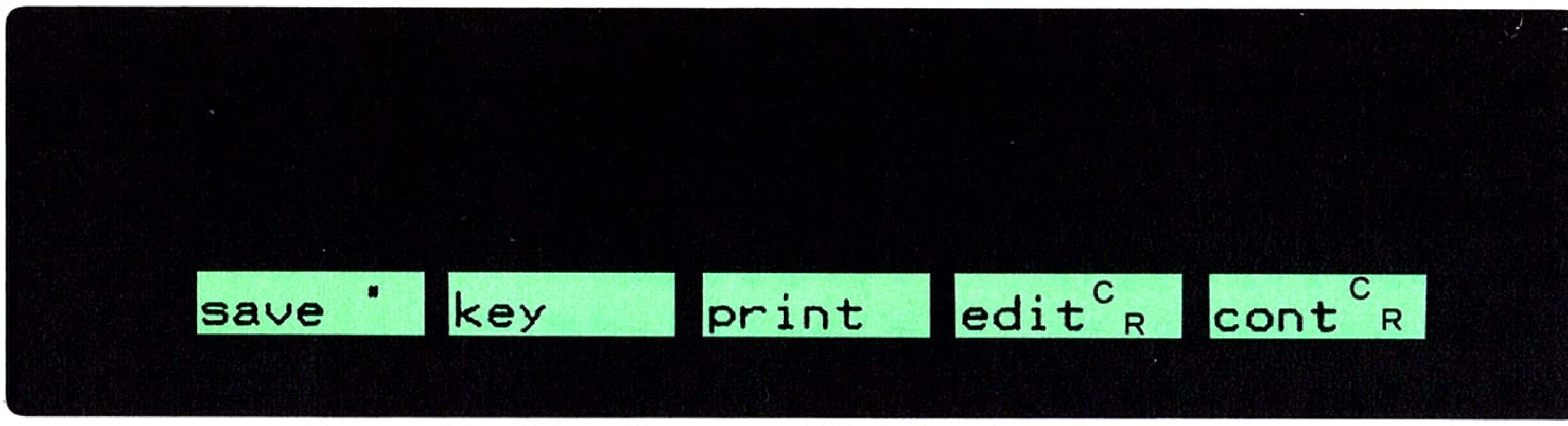
キーボードの一番上に f･1 から f･5 までのキーがあります。

これをファンクションキーといいます。

ファンクションキー

これらのキーには電源を入れたとき、ある一定のことが定義されています。3ページ、4ページの画面の一番下に細長い四角が5つあって、その中に文字が表示されていますね。これが、左から f･1 ～ f･5 キーに対応した定義内容です。

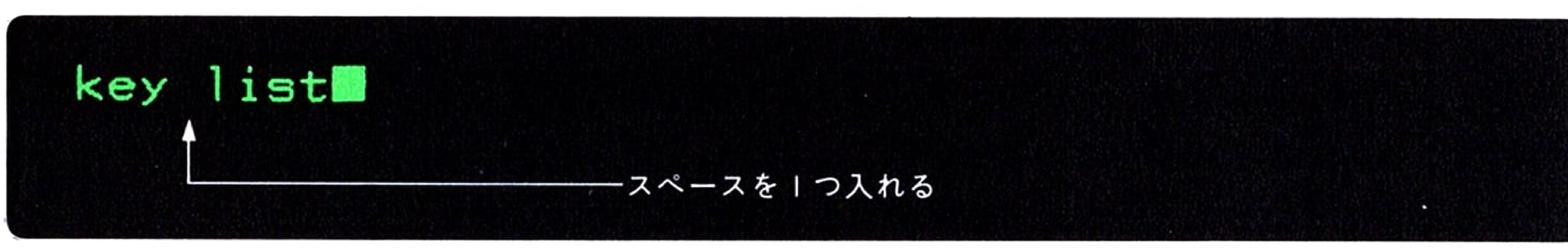
それでは、(SHIFT) キーを押してみてください。表示されている文字が次のように変わりました。



これらは(f・6)～(f・10)の定義内容です。キーボードには (f・6) から (f・10) までのキーは実際にはありませんが (SHIFT) ＋ (f・1) を (f・6) に、(SHIFT) ＋ (f・2) を (f・7) に、というように対応させています。

これらの定義内容についてはいまわからなくてかまいませんが、プログラミングをするときによく使う命令（コマンド）が定義されている、と考えてください。

ここで、ホームクリアキーで画面をキレイにして、次のようにキーインしてみてください。小文字でも大文字でもかまいません。



スペースキー

●本書ではスペースキーは (スペース) のように表示する場合があります。

key のyのうしろにスペースを1つ入れますが、(→) キーを使っても、キーボードの一番下にある細長い棒状のキー（スペースキーといいます）を使ってもかまいませんが、できるだけスペースはスペースキーを使って入れる習慣をつけましょう。

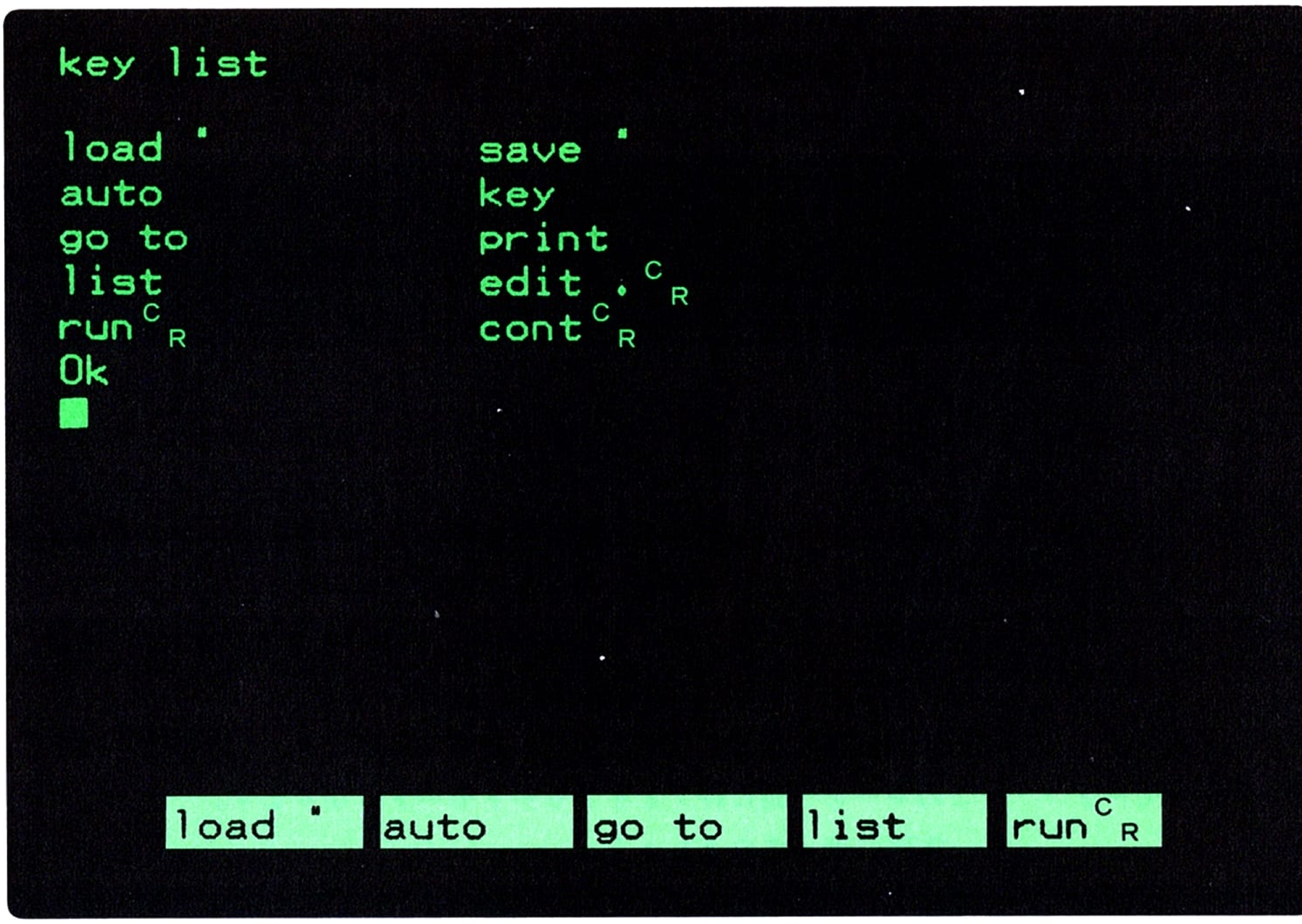
上の図の表示のようにできましたら、(↵) キーを押してみてください。画面は次ページのようになりました。

ファンクションキーの定義内容が画面に表示されました。左側が、(f・1) から (f・5) に、右側が (f・6) から (f・10) に対応しています。「r u n」などのうしろに「$^{C}{}_{R}$」というのがあります。これは

キャリッジリターン

「キャリッジリターン」と読みますが、(↵) キーを押すのと同じ機能を持っています。

ファンクションキーについての説明はいまはこれで終りにして、あとはプログラミングの過程で実際に使っていくことにします。解説を加えるよりは、そのほうがわかりやすいと思うからです。



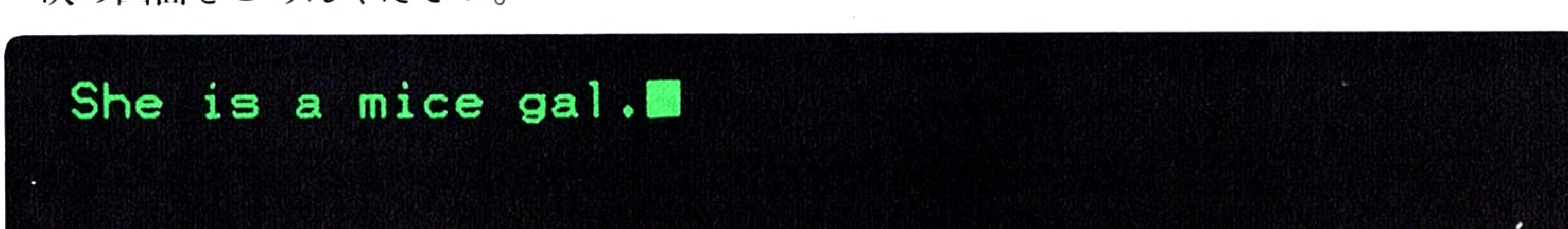
グラフィックキャラクタ

（カナ）キーの右どなりに（GRPH）キーがあります。これがグラフィックキャラクタ表示のためのキーです。

Ⅷページの下の図をごらんください。緑色の部分がグラフィックシンボルキーです。（GRPH）キーを押しながらこれらのキーを押すと、対応するグラフィックキャラクタが画面に表示されます。みなさん各自でためしてみてください。

文字を修正したいとき

次の画面をごらんください。



「あの子はいいこだ」といいたいところ、「nice」のつづりが間違えて「mice」になってしまったため、なんだか「ネズミむすめ」みたいになってしまいました。これではいけません。さっそくなおしまし

ょう。修正は簡単です。

カーソル移動キーを使ってカーソルを「m」の上に重ねます。そして N キーを押せばよいのです。

```
She is a mice gal.
```

カーソルをここに移動して N を押す

デリートとインサート

大きいほうの ⏎ キーの上に「INS」「DEL」と書いたキーがあります。これが INS キーと DEL キーです。

INS は「インサート」、つまり文字を挿入するときに、DEL は「デリート」、つまり削除するときに使います。

デリートモード

インサートモード

このままの状態で押すとデリートモードになり、SHIFT キーとこのキーを一緒に押すとインサートモードになります。

実際に使ってみましょう。

```
NEEC
```

上の画面ではEが1つ多いようです。これを削除するには、カーソルを2番目のE、あるいはCの上に重ね、DEL キーを一度たたきます。

```
NEEC
```

この状態でデリートキーを押す

次はインサートモードです。「NC」のNとCの間にEを入れて、「NEC」としてみましょう。まず、Cの上にカーソルを重ね、ここで SHIFT キーと INS キーを一緒に押します。するとインサートモードになりますから、E キーをたたいてください。

```
NC
```

カーソルをCに重ねインサートモードにし、キャピタルロックして E キーを押す

インサートモードから抜けでるには、どれでもいいですからカーソル移動キーを押してください。インサートモードにしてあると、何文字でも挿入できます。

1・3　システムディスクをコピーする

この節はディスク内蔵タイプの機種をお持ちの方が対象です。モデル10をご使用の方は、第２章へお進みください。

システムディスクのコピーを作る

１・１節でふれた、ラベルに「N88DISK-BASIC」と大きく書かれたほうのディスクをシステムディスクといいます。

カタカナ用語がでてきてちょっとわかりにくいかもしれませんが、システムディスクには、次のようなものがおさめられています。

❶ **システム**――ディスクを起動させる原動力となるもの

❷ **ユーティリティプログラム**――ディスクにおさめられているプログラムとかデータを新しいディスクにそのままコピーしたりできる便利なプログラム

❸ **デモプログラム**――ＰＣ－8801ｍＫⅡのコマンド、ステートメントをフルに活用したデモンストレーションプログラム

❹ **チューター**――キーボードの使い方とかグラフィックスの考え方が学べる、いわば家庭教師的なプログラム

❺ **拡張命令マシン語プログラム**――タートルグラフィックとかサウンド機能を使えるようにするためのプログラム

❻ **各種データ**――デモンストレーションの画面作りに必要なグラフィックデータ

といわれても、いまはなかなか理解できないかもしれませんが、この本を読み進んでいくうちに除々にわかってくると思いますので、そのときに再びここへ戻ってみてください。

ここで扱うのは❷のユーティリティプログラムです。

いまあなたが持っているシステムディスクは、ディスクを起動させる非常に大切なものです。何らかの手違いでこわれることもありえます。万一の場合にそなえて、このシステムディスクとまったく同じものを作っておきましょう。つまり、システムディスクをコピーしてしまうのです。

この節ででてくる命令は、まだ説明していませんのでわからないかもしれませんが、書いてあるとおりにキーインしていってください。また、ここではモデル30を対象に説明を進めますので、モデル20の方は、要領はだいたい同じですから、まずこの記事を読んでから実行し

てみてください。

なお、ここでは新しいディスクを2枚必要としますので「5インチ両面倍密度版」（ディスク内蔵型の場合）を準備してください。

それでは始めますが、ディスクは起動していますか？　まだの人はシステムを立ち上げてください（ディスクを起動することを「立ち上げる」といいます）。

まず、ホームクリアキーで画面をクリアしておきましょう。そして次のようにキーインしてください。

FILES(ファイルズ)

F I L E S (スペース) 1 ⏎

キーインするアルファベットは、ことわりがない限り、大文字でも小文字でもかまいません。

また、⏎ はリターンキーを押すことを意味します。

さて、いまのキーイン操作はドライブ1に入っているディスク（現在はシステムディスクが入っています）の中におさめられているファイル（プログラムその他）をすべて表示する命令です。〈注〉

〈注〉ドライブ1のディスクのファイル名を表示させる場合、上のキー入力で「1」というドライブ番号は省略してもかまいません。

ドライブ2側のディスクのファイル名を表示させたいときは、「1」を「2」に変えればよいのです。現在はドライブ2側にディスクをセットしてありませんから、実行しないでいいですよ。

画面は次のようになりましたか？

```
files 1
demo   .n88 7  tutor .n88 12 dskut1.n88 4  dskut2.n88 9  ngen  .n88 13
@load .n88 1  @exst *    2  demoex.n88 3  *     *   1  *demo0*    4
*demo1*    1  *demo2*    1  *tutor*    1  $pict0 dat 4  $pict1 dat 4
$pict2 dat 6  $pict3 dat 6  $pict4 dat 10 $pict5 dat 12 $pict6 dat 12
$pict7 dat 14 $pict8 dat 14
Ok
```

ずいぶんいろいろな欧文の名前が表示されましたね。この中の1つ1つ、たとえば左上スミの「demo. n88」などをファイル名といいます。そして、そのうしろの数字「7」は、このファイルがディスク上で占有している容量を示しています。

ファイルにはそれぞれ意味がありますが、いまはこのくらいの知識でよいでしょう。

画面をもう一度ごらんください。一番上の行に「dskut1. n88」というファイルがありますね。そしてそのとなりに「dskut2. n88」があり

ます。この2つがディスクユーティリティプログラムです。おそらく「disk utility」からファイル名を作ったのだと思います。

ファイル名に「1」という数字が入っているほうは1ドライブのモデル20型のための、「2」が書かれているほうはモデル30型のためのユーティリティプログラムです。

ここでは、モデル30型を対象に話を進めますので、後者を使うことになります。

このプログラムをディスクからコンピュータのメモリ（記憶領域）に呼び出して、これを実行することによりシステムディスクのコピーを作ります。

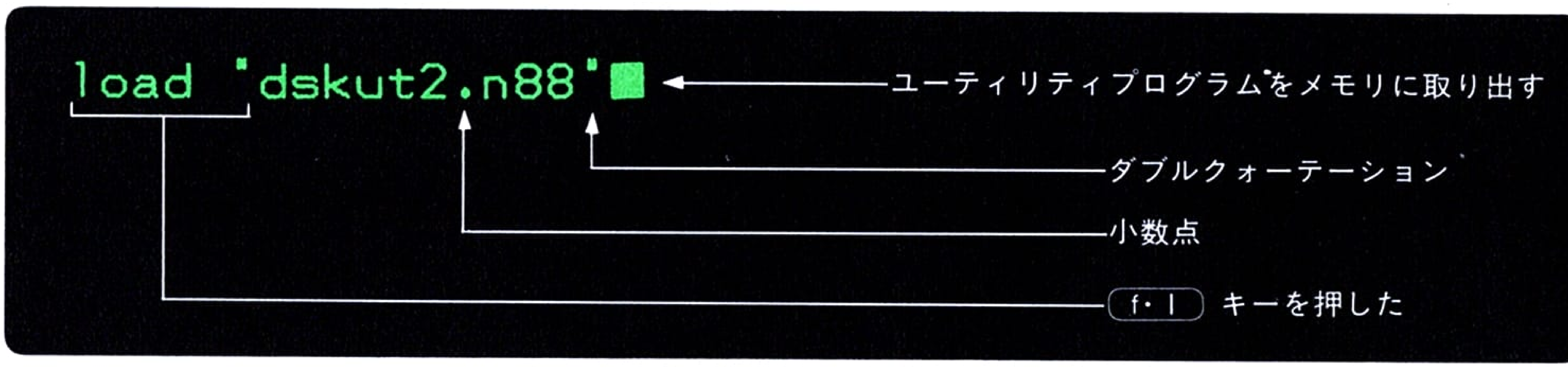
理屈は抜きにして説明するとおりにキーインしてください。

次のようにキーをたたいてください。

| f・1 | d | s | k | u | t | 2 | ・ | n | 8 | 8 |

まだリターンキーは押さないでください。上のキー表示にあるように、この場合アルファベットはすべて小文字で入力してください。

画面は次のようになっていますか？



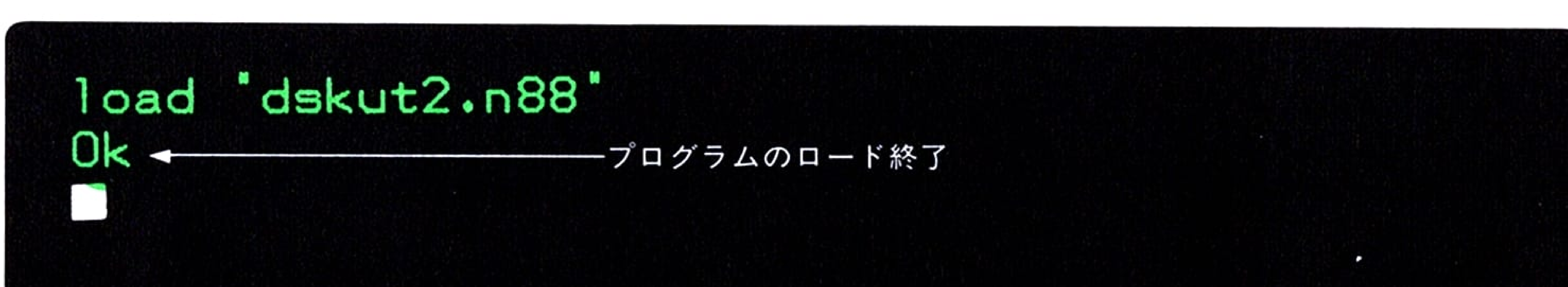
この場合、ファイル名は大文字ではダメですよ。

間違いなかったら ⏎ キーを押してください。すると、ドライブ1側のアクセス表示用ＬＥＤが赤く点灯し、ディスク上のプログラムがメモリへ移されます。

〈注〉4ページの画面をごらんください。f・1 キーに「load "」が定義されていましたね。

「ＬＯＡＤ」とは、「装てんする」などという意味があるように、プログラムをメモリへ運び込むことです。

しばらくしてＬＯＡＤコマンドが終了し、画面には「Ok」が表示されます。



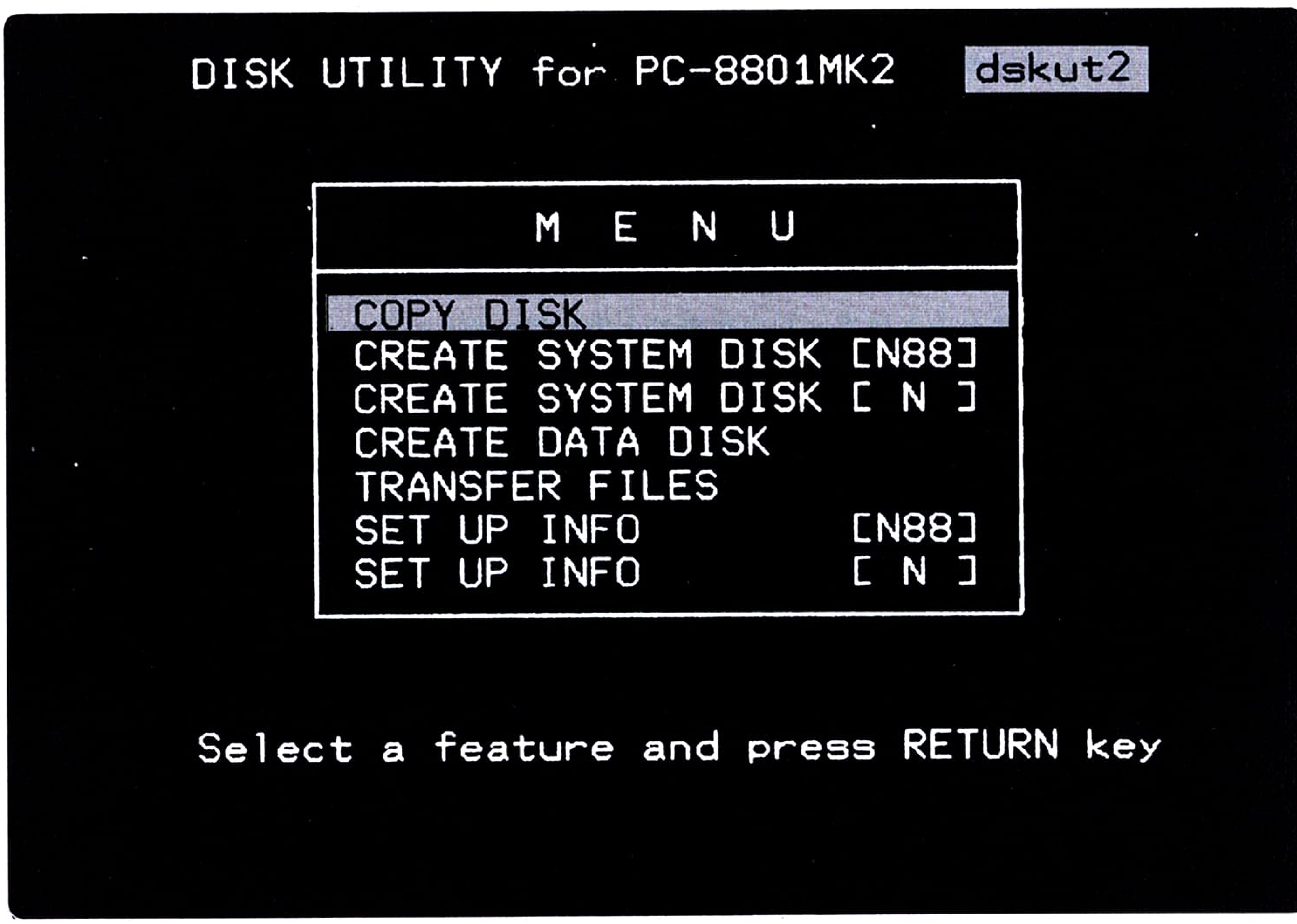
さあ、これでユーティリティプログラムをロードできました。あとは、これを実行すればよいのです。

プログラムを実行するにはRUN（ラン）コマンド（プログラムを走らせるという意味）を使い次のようにキーインするのですが、これとまったく同じ内容が f・5 キーに定義されていますので、こちらを使うことにします。

RUN ⏎

これと同じことが f・5 キーに定義されている

さあ、それでは f・5 キーを押してください。次のようなメニュー画面が表示されましたね。



表の中にプログラムメニューが7つありますね。そして現在、一番上の「COPY DISK」が白い枠で囲まれて文字が反転しています。この白い枠はカーソル移動キー ↑ ↓ を使って希望するメニューのところへ移動することができます。ちょっとためしてみてください。どうですか？ 移動したでしょう。

私たちがいま必要としているのは「COPY DISK」プログラムですから、もう一度白い枠をこの上に重ね、リターンキーを押してください。

すると画面に次のようなメッセージが表示されます。意味は、「マスターディスクをドライブにセットしてください。セットしたドライブ番号は何番ですか？」ということです。

```
[[[ COPY DISK ]]]

Mount MASTER disk on drive.
Drive number? ■
```

[[[COPY DISK]]] ← プログラムの見出し
Mount MASTER disk on drive. ← マスターディスクをドライブにセットしてください
Drive number? ← ドライブ番号は？

私たちの場合にはシステムディスクがマスターディスクになります。ドライブ1にセットしてありますから、 [1] キーを押して ↵ です。

```
1 ↵
```

すると、新しいディスクをドライブにセットするように指示してきます。そして、やはりドライブ番号をたずねています。

```
Mount NEW disk on drive
Drive number? ■
```

用意してある新しいディスクをドライブ2にセットしてリターンキーを押してください。

```
2 ↵
```

今度は物理フォーマットをするかどうか聞いてきます。

```
Do you need physical formatting(y/n)? ■
```

物理フォーマットとは、市販されている5インチのフロッピィディスクはそのまま使うことができないので、使えるように初期化することをいいます。これ以上の説明は省略しますが、とにかく物理フォーマットという作業をしないと使用できない、と覚えておいてください。

ここでは次のようにキーインしましょう。

```
y ↵
```

すると画面に次のような表示が現れ、物理フォーマット実行の確認を求めています。

```
Format a disk on drive 2
Sure(y/n)? ■
```

やはり「y」をキーインします。

```
y ↵
```

「物理フォーマット中」のサインが次のように表示され、ディスクの初期化（イニシャライズ）が行なわれます。

```
Physical formatting.
```

しばらくして物理フォーマットが終了すると、いよいよマスターディスクに書き込まれている情報を新しいディスクへコピーする作業が開始されます。最後に次のような画面になってコピーが終了します。

```
Copying track 77
Copying track 78
Copying track 79

Completed.
Ok
█
```

さあ、これでシステムディスクを新しいディスクにコピーすることができたわけですが、うまくいったかどうか確かめてみましょう。

確認の方法はＦＩＬＥＳコマンドを使って、ドライブ１のディスクとドライブ２のディスクのファイル名を表示させてみればよいですね。

次のようにキーインしてください。

```
files ↵
```
← ドライブ１の場合ドライブ番号は省略できる

```
files 2 ↵
```

画面は次のようになりました。

```
files ←――――――――――――――――――――ドライブ１のディスクのファイル名を表示
demo   .n88 7  tutor .n88 12 dskut1.n88 4  dskut2.n88 9  ngen   .n88 13
@load  .n88 1  @exst *     2  demoex.n88 3  *      *   1  *demo0*     4
*demo1*     1  *demo2*     1  *tutor*     1  $pict0 dat 4  $pict1 dat 4
$pict2 dat 6  $pict3 dat 6  $pict4 dat 10 $pict5 dat 12 $pict6 dat 12
$pict7 dat 14 $pict8 dat 14
Ok
files 2 ←――――――――――――――――――ドライブ２のディスクのファイル名を表示
demo   .n88 7  tutor .n88 12 dskut1.n88 4  dskut2.n88 9  ngen   .n88 13
@load  .n88 1  @exst *     2  demoex.n88 3  *      *   1  *demo0*     4
*demo1*     1  *demo2*     1  *tutor*     1  $pict0 dat 4  $pict1 dat 4
$pict2 dat 6  $pict3 dat 6  $pict4 dat 10 $pict5 dat 12 $pict6 dat 12
$pict7 dat 14 $pict8 dat 14
Ok
█
```

どうですか？　ドライブ１のディスクの内容も、ドライブ２のディスクの内容もまったく同じですね。つまり、コピーに成功したのです。このコピー版のディスクには当然システムもコピーされていますから、これを使ってディスクを起動させることができるわけです。

そこで、オリジナルのシステムディスクはどこか安全な場所に保管しておき、これからは、このコピーしたシステムディスクを使うことにしましょう。

では、ドライブ1のシステムディスクを取り出し、ドライブ2のディスクをドライブ1に入れてください。

システムだけをコピーする

14ページの画面を見てもわかるように、システムディスクにはいろいろなファイルが保存されていますが、この中には普段必要ないものがたくさんあります。グラフィックデータなどでずいぶんディスクのメモリを浪費しています。ですから、ここではディスクを立ち上げるシステム（システムはファイルではありませんよ。ディスクを起動させるための情報としてディスクのある位置に書き込まれています）だけを2つ目の新しいディスクにコピーすることにしましょう。

ユーティリティプログラムはまだメモリにありますから、これを、f･5 キーを押して実行しましょう。f･5 キーを押してください。すると16ページのようにメニューが表示されますから、⬇ キーを一度押して白枠を上から2つ目のメニュー「CREATE SYSTEM DISK 〔N88〕」のところへもっていってください。

そしてリターンキーを押します。画面は次のようになりました。

```
[[[ CREATE SYSTEM DISK ]]]

      N88 DISK-BASIC

Mount SYSTEM disk on drive.
Drive number? ■
```

システムディスクはドライブ1にセットしてありますから、

```
1 ↵
```

でよいですね。

次は、新しいディスクをドライブにセットするよう指示してきます。

```
Mount NEW disk on drive.
Drive number? ■
```

2枚目の新しいディスクをドライブ2にセットしてください。セットできたら 2 を押してリターンです。そして物理フォーマットの意

志を問うメッセージに「y」を入力すると、

```
Format a disk on drive 2
Sure(y/n)? ■
```

のように確認を求めてきますからここでも「y」を入力しましょう。

```
y ↵
```

画面に次のメッセージが表示され物理フォーマットが行なわれます。

```
Physical formatting.
```

続いてシステムフォーマットが行なわれます。

```
System formatting.
```

そして最後にシステムだけをコピーして終了です。

```
Copying system.

Completed.
Ok
■
```

〈注〉電源を切るときは、すべてのディスクをドライブから取り出してからにしてください。

これで作業はすべて終りました。新しいディスクにシステムがコピーされたかどうかためしてみてください。ドライブ2のディスクを取り出してから電源を切り、これをドライブ1にセットして再び電源を投入してみてください。4ページと同じようにディスクが起動すると思います。

このシステムだけをコピーしたディスクは、この本を終えて次のステップへ進むときに必要になりますから、それまでオリジナルのシステムディスクとともに大切に保管しておいてください。

デモプログラムの実行

みなさん、ちょっと疲れたのではありませんか? 骨休めにデモプログラムを楽しんでみてください。コピーしたシステムディスクをドライブ1に入れ、次のようにキーインします。

```
run "demo.n88" ↵
```

また、もう少し勉強したい方は次のようにどうぞ。

```
run "tutor.n88" ↵
```

第2章

パソコンで何ができるの?

▼▼

2・1　四則演算をやってみよう

〈注〉前節のユーティリティプログラムを実行したのちは、画面の文字が大きくなったと感じませんか？　これは、画面モードが電源投入時のそれと異なるモードになったためで、別に支障ありません。画面モードの切り換えについては後で説明します。

キーボードからのアルファベット、カナ、数字、記号などのキーイン、いかがでしたか？　最初のうちはどのキーがどこにあるかちょっとまごつきますが、そのうち慣れてくると思います。しかし、カタカナは、あまり使う機会がないせいか、いつになっても手間どるようです。

さて、せっかくのコンピュータ、カーソルを動かしたり文字を打ち込んだりだけではしかたありませんね。「なんでもできる」とうわさでは聞いていますが、いったい何ができるのでしょう。

私たちはいままで計算を電卓とか筆算でやっていました。ひとつコンピュータにこれをやらせてみようではありませんか。

まずはたし算から

私たちが電卓でたし算をする場合、次のようにキー入力していました。たとえば、5＋3の場合、

[5] [+] [3] [=]

このように入力すると、小さな画面に答が 8と表示されました。コンピュータでの演算のいとぐちをつかむために、この辺の計算からやってみましょう。

まずホームクリアキーで画面をクリアしてから、電卓の場合と同じようにキーをたたいてみてください。

画面は次のようになり、イコールの右でカーソルが点滅(ブリンク)しています。

```
5+3=■
```

おや、答がでると思ったらでてきません。どうしたのでしょう。

パソコンでは、ある一定のことを入力したら [↵]、つまり**リターンキーを押す**ことになっているのです。押してみましょう。

```
5+3=
■
```

やはりダメでした。ウンともスンともいいませんね。どうしたらいいでしょうか。

実はパソコンでは、この場合イコールは使わずに、PRINT（プリント）という命令を使うのです。プリント、うーん、何かわかるような気がしませんか？　答を画面にプリントする、つまり表示するのです。そして構文は、「プリントせよ、○○を」というように英語式にＰＲＩＮＴを先頭にもってくるのです。

PRINT

それではさっそくキーインしてみましょう。大文字でも小文字でもかまいませんよ。ＰＲＩＮＴのＴのうしろには一応スペースを入れておきましょう。スペースキーを一度たたけばよかったですね。

```
PRINT 5+3■
```

カーソルはまだここにある

スペースを１つ入れる

問違いなくキーインできましたか？　まだカーソルは３のうしろでブリンクしています。

ここでリターンキーを押してみてください。画面はあッというまに次のようになりました。

```
PRINT 5+3
 8
Ok
■
```

やっと正解が得られました。

ここで、もう少し大きな数で演算をしてみますが、その前にプリントについて一言。

いまみなさんはプリントを P R I N T とキーインしましたが、「？」または f・8 キーで代用することができます。

？（クエッションマーク）

「？」は SHIFT キーと「？」の書かれたキーと一緒に押せばよいですし、f・8 キーというのは SHIFT キーと f・3 キーを一緒に押すのでしたね。18ページのファンクションキー表示を見てください。f・8 キーに print が定義されています。

ここでは「？」を使ってみましょう。

```
? 115+2282
 2397
Ok
■
```

クエッションマークでＰＲＩＮＴの代用ができるのですから、このほうが楽でいいですね。また（f・8）キーも、使ってみるとけっこう便利ですよ。

もっとも、最初はＰＲＩＮＴと１字１字入力してキーに慣れたほうがよいかもしれません。まあ、とにかくあなたの趣味におまかせします。使いやすいほうをお使いください。

たし算はできました。次は、ひき算です。

ひき算

ひき算もたし算と原理はまったく同じです。ＰＲＩＮＴとキーインし、スペースキーを１回たたき、次に算式をキーインします。そしてリターンキーを押します。この、リターンキーを押すのを忘れないでください。やってみましょう。

```
? 1000-1200
-200
Ok
█
```

上の画面では、ひく数のほうが大きかったので答はマイナスになりました。

答の表示位置は「？」の真下から右へ －200 となっています。

たし算の場合はどうだったでしょう。前ページの真中の画面を見てください。答の８という数字は「？」の真下に表示されず、右へ１つずれています。「？」の下にスペースが１つ入っていますね。このスペースはいったい何でしょうか。

８という数は「＋８」と同じですね。上の答の場合にはマイナス符号がちゃんとついていましたが、結果がプラスのときにはプラス符号は表示されませんでした。

もうおわかりですね。答が正の場合にもプラス符号はあるのですが、ただそれが表示されず、スペースとなって現れているのです。

演算結果がプラスの場合には必ず、プラスの符号分として１字分とられているのです。このことは大切ですから頭のすみに入れておいてください。

それでは、たし算とひき算を自分でいくつか練習してみてください。

かけ算とわり算

かけ算とわり算も演算記号が違うだけですから、パソコンへの命令のしかたは同じです。

ただし、数学では、かけ算は「×」、わり算は「÷」を使っていましたが、パソコンではそれぞれ、「／」（スラッシュ）と「＊」（アスタリスク）を使います。

スラッシュ
アスタリスク

記号を変えればいいだけですから、簡単ですね。スラッシュとアスタリスクはテンキー側とメインキー側に各1つずつあります。見つかりましたか？

次の計算をやってみてください。

```
? 2*3
 6
Ok
? 2/3
 .666667
Ok
■
```

上のような結果になりましたか？

●小数の場合、整数部分の0は省略されます。

わり算の答は小数点以下6桁までしか表示されていませんね。もう少しくわしく答をだしたいときは、2のうしろに「#」（シャープ）をつけて次のようにします。

シャープ

```
? 2#/3
 .6666666666666667
Ok
■
```

だいぶくわしい結果が得られました。小数点以下16桁の精度です。

パソコンでの整数と実数

整数というのはごぞんじですね。1、2、3……というように、きりのいい数です。数学では相当桁数の多い数でも小数点以下がなければ整数として扱いました。

ところが、パソコンでは制限があり、－32768～＋32767の範囲の整数を「パソコンでの整数」として扱います。

整数

ですから、この範囲外、あるいはこの範囲内であっても小数点以下

があると実数として扱われます。

パソコンの実数には2つの型があります。1つは単精度型、もう1つは倍精度型です。

単精度型実数

単精度型は前ページの演算結果0.666667のように、有効数字7桁の精度で6桁以下で表示されます。

倍精度型実数

一方、倍精度型は、シャープをつけた演算結果のように、有効数字17桁の精度で16桁以下で表示されます。

整数型、単精度型、倍精度型を強調したいときは、次のような記号を数値のうしろにつけることになっています。

- **整数の場合———％（パーセント）**
- **単精度型実数の場合———！（イクスクラメーションマーク）**
- **倍精度型実数の場合———＃（シャープ）**

少々頭がいたくなりそうなテーマですが、いまは、こんなことがある程度で流してください。

加減乗除の応用

次のような計算式はどうしたらよいでしょうか。

$$330 \times \left[\left(43 - \frac{4}{5} \right) \times 3 + 21 \right]$$

●小中学生の数学の教科書の数式の問題をやってみてください。キー操作の勉強になりますよ。

応用といっても、これも簡単です。「ＰＲＩＮＴ」または「？」をキーインし、続いて上の通りに打ち込んでいけばよいのです。

ただし、パソコンでは「かぎカッコ」も「まるカッコ」も、同じまるカッコを使います。まるカッコはどこにあるかわかりますね。また、カッコの向きを間違わないでください。

次のようになれば正解です。

```
? 330*((43-4/5)*3+21)
 48708
Ok
█
```

2·2　関数計算もできます

みなさんごぞんじの平方根とか三角関数の計算も、コンピュータは忠実にやってくれます。

平方根の計算

ＳＱＲ関数

パソコンには平方根記号（$\sqrt{\ }$）がありませんので、ＳＱＲ（スクェア）関数を使います。たとえば$\sqrt{5}$の計算をするときは、次のようにキーインし、リターンキーを押します。ＰＲＩＮＴのかわりに「？」を使ってもいいですよ。

```
PRINT SQR(5)
 2.23607
Ok
■
```

ずいぶん簡単にできてしまいますね。

それでは次に$\sqrt{7}$の計算をしてみましょう。

$\sqrt{5}$の場合と同じように、いちいち

[P] [R] [I] [N] [T] [スペース] [S] [Q] [R] [(] [7] [)] [↵]

と入力してもよいのですが、もう少し楽な方法があります。

まずカーソルを動かして、上の画面「５」のところへもっていってください。

```
PRINT SQR(5)
 2.23607
Ok
```

（「5」に）カーソルをここに重ねる

できましたか？　そうしたら、５を７に変えてリターンキーを押します。

どうですか？　答がでてきたでしょう。

```
PRINT SQR(7)
 2.64575
Ok
■
```

同じように$\sqrt{22}$の計算をしてみましょう。

まずカーソルを「7」の上に重ね、[2]キーを押して7を2に変えます。するとカーソルはカッコの位置へ移り点滅しています。

```
PRINT SQR(2)
 2.64575
Ok
■
```

ここで [2] [)] [↵] とキーインしてもいいのですが、練習のためインサートモードでやってみましょう。

(SHIFT) キーを押しながら (INS) キーを一度押してインサートモードにします。そうしましたら、[2] キーを押し [↵] です。

インサートモードはこれからもよく使いますので、できるだけ使って慣れておきましょう。

三角関数の計算

三角関数はグラフィックスではよく使いますが、ふつうはそれほど必要ありません。

SIN

計算方法ですが、たとえばSIN30°の場合、

```
? SIN(30) ↵
```

と入力しても、おかしな答がでてきてうまくいきません。

ラジアン

三角関数では単位を、度ではなくラジアンを使用します。

180°がπラジアンですから、任意の角度Xをラジアンに換算するには、

$$X \times \frac{\pi}{180} \quad (\text{ただし、}\pi = 3.14159)$$

とすればよいわけです。

SIN30°の計算をやってみましょう。

```
? SIN(30*3.14159/180)
 .5
Ok
■
```

コサイン（COS）、タンジェント（TAN）も同じようにできます。それぞれ各自でためしてみてください。

2・3　変数を使ってみよう

変数といっても、別に「ヘンな数」ではありませんよ。これは数学でいう VARIABLE QUANTITIES で、変化しうる数とでもいうのでしょうか。

まあ、そんなことはどうでもいいですから、23ページでやった「5＋3」を変数を使って計算してみましょう。

新しいテーマにチャレンジするのです。画面もクリアしてからスタートしましょう。

まず次のようにキーインしてください。

```
X=5 ⏎
```

⏎ はリターンキーを押すことですよ。

できましたら同じように、

```
Y=3 ⏎
```

とキーインします。Yは小文字でもかまいませんよ。

いいですか？　これでXが5になり、Yが3になったわけです。そして、いま使ったXとYのことを**変数**といいます。つまり、Xという変数に5が入り、Yという変数に3が入ったのです。

変数

●「変数とは器のようなものだ」と考えると理解がはやいかもしれません。

コールドスタート

私たちが使うことのできるコンピュータの中の記憶領域（メモリ）を全部キレイにして（クリアして）、最初からスタートすることを**コールドスタート**といいます。

コールドスタート

ですから、電源を入れたときはいつもコールドスタートのわけです。しかし、プログラミングの途中でいちいち電源を切って、ふたたびONにするというのもわずらわしいですね。そこで登場するのが**リセット（RESET）ボタン**です。本体前面の右の扉を開けると、一番左に白いボタンがありますが、これがそうです。

リセットボタン

それではリセットボタンを押してみましょう。ディスク内蔵タイプの人は、ディスクがセットされていて、挿入口のフタがちゃんと閉めてあるかどうか確認してください。

さあ、それでは押してみましょう。2ページ、4ページでやったように、使用するファイル数を問うメッセージが表示されました。これから先の要領はもうおわかりですね。そうです、ディスクを使わない人

は ⏎ キー、使っている人は 5 ⏎ でしたね。
これで準備ＯＫです。メモリはすっかりクリアされました。

変数を使って計算する

それでは最初のテーマに移りましょう。
次のようにキーインしてみてください。

```
? X
```

キーインできたらリターンキーを押します。画面は次のようになりました。

```
? X ←──── Xの中味を表示せよ
 0
Ok
■
```

「？　X」、つまり**「Xという変数の中味・内容を表示せよ」**と命令してリターンキーを押したところ、ゼロという結果がかえってきました。Ｙについてもためしてください。おそらく同じ結果が得られると思います。

このように、**コールドスタートをしたときは、すべての変数はゼロにクリアされているのです。**

次に前ページの「Ｘ＝５ ⏎ 」「Ｙ＝３ ⏎ 」の命令をもう一度入力してみましょう。次のようになりましたか？

```
X=5
Ok
Y=3
Ok
■
```

先ほど説明しましたように、これでＸという変数に５が、Ｙという変数に３が入ったわけです。確認してみましょう。

コールドスタートしたときゼロだった変数ＸとＹに、果たして５と３が入っているかどうか……。次のようにキーインすればいいですね。

```
? X ⏎
? Y ⏎
```

画面は次のようになります。

```
? X
 5
Ok
? Y
 3
Ok
■
```

どうですか？　初め０だったのが、それぞれ５、３と変化しましたね。これが変数です。

それでは「５＋３」を、この変数を使って計算してみましょう。

Ｘには５が、Ｙには３がすでに入っていますから、ＸとＹの内容を加えれば結果が得られますね。「ＸとＹの中味をたして、その答を表示せよ」と命令すればよいわけです。キーインしてみてください。次のような画面になりましたか？

```
? X+Y
 8
Ok
■
```

めでたし、めでたしでした。

代入文

ここで新しい変数Ｚに登場してもらいましょう。この変数は新人ですから、最初は当然ゼロになっているはずです。

```
? Z
 0
Ok
■
```

この変数Ｚさんには、変数Ｘと変数Ｙの中味をたしたものを入れることにしましょう。次のようにキーインします。

```
Z=X+Y ⏎
```

この等式の意味ですが、数学では「Ｚの値とＸ＋Ｙの値が等しい」と考えますが、コンピュータでは「Ｘの中味とＹの中味をたして、それをＺに入れる（代入する）」ということになります。そして、この

等式を代入文といいます。

ですから先ほどの「X＝5」にしても「Y＝3」にしても一種の代入文で、「5という値を変数Xに代入する」「3という値を変数Yに代入する」と考えたほうがよいわけです。

これから先、代入文はひんぱんに使いますが、数学の等式というよりは、いま説明したような考え方をするように習慣づけましょう。

さあ、これで、ZにXとYの和が代入されたはずです。確かめておきましょう。

```
? Z
 8
Ok
■
```

マルチステートメント

ステートメント

＊セミコロン（;）と間違わないようにしてください

前項では、「X＝5」とか「Y＝3」とか「Z＝X＋Y」をそれぞれキーインしては ⏎ キーを押していましたが、この1つ1つを**ステートメント**といいます。そして、これらのステートメントは1つにまとめることができます。**コロン（：）**＊を使って次のように表現します。

```
X=5:Y=3:? X+Y
```

- X=5 ― 変数Xに5を代入せよ
- Y=3 ― 変数Yに3を代入せよ
- ? X+Y ― 変数Xと変数Yの和を表示せよ

マルチステートメント

このように複数のステートメントで構成されるステートメントを**マルチステートメント**といいます。これもよく使いますから覚えておいてください。

このマルチステートメントを使って、いまの計算をしてみましょう。上のようにキーインし、リターンキーを押します。

```
X=5:Y=3:? X+Y
 8
Ok
■
```

コンピュータはこのステートメントが入力されリターンキーが押されると、最初のステートメントから忠実に実行していきます。まず、

Xに５が代入され、次にYに３が代入され、そしてXとYの内容を計算して、その結果を表示する、というステップをふみます。

次に、変数の値を変えてためしてみます。

X＝150、Y＝120としてみましょう。次のように入力してもよいのですが、ここでは前のマンチステートメントを利用することにします。

```
X=150:Y=120:? X+Y ↵
```

インサートモードを使います。まず、カーソルを「５」に重ね [1] のキーを押します。するとカーソルはコロン(：)の上にいきますから、ここで (SHIFT) キーを押しながら (INS) キーを一回押してインサートモードにします。両方のキーから手ははなしていいですよ。

そして [5] [0] の順でキーインします。

次に [→] キーを使って「３」にカーソルを重ね、同じ手順でYの値を変えます。

画面は次のようになっていますか？

```
X=150:Y=120:? X+Y
 8
Ok
```

カーソルがここでブリンクしている

上の画面のようになりましたら、リターンキーを押してください。カーソルをYのうしろまでもっていく必要はありませんよ。次のように正解が表示されました。

```
X=150:Y=120:? X+Y
 270
Ok
█
```

このように、マルチステートメントをいちいちキーインしなくても、X、Yの値さえ変えれば、何度でも計算することができます。

ここで、変数の値は変更するごとに変わっていることに注意してください。現在はXの内容は150、Yの内容は120、前ページで使った変数Zの内容は８です。X、Yの値を変えてからZ＝X＋Yを実行していませんから、Zの中味は変化していません。

2・4　画面に文字を表示する

前節で「?　X＋Y」というステートメントがありました。これは変数Xと変数Yの和を表示せよという命令でしたね。

それでは「X＋Y」という文字そのものを画面に表示するにはどうしたらよいでしょうか。答を先に示します。

文字列(ストリング)

ダブルクォーテーションマーク

次のように、表示したい文字列（**ストリング**といいます）を**ダブルクォーテーションマーク**でくくって、その前にＰＲＩＮＴ（？でもかまいません）をつけます。

```
PRINT "X+Y"
```

ダブルクォーテーションマークでくくる

実際にやってみましょう。

```
PRINT "X+Y"
X+Y
Ok
■
```

文字列の代入文

文字型変数（ストリング変数）

$（ダラー）

数値変数と同じように、文字列も変数に代入することができます。この場合の変数は数値型変数ではなく**文字型変数（ストリング変数）**といい、二者を区別するために**$**（ダラーまたはドル）マークを変数のうしろにつけます。

```
A$="X+Y"
```

●代入できる文字数は255文字までです。

これでＡ＄（エーダラーと読みます）という変数に「Ｘ＋Ｙ」が代入、つまりＡ＄の中味がＸ＋Ｙになったわけです。

```
A$="X+Y"
Ok
? A$
X+Y
Ok
■
```

各自、いろいろためしてみてください。

2・5 変数について

パソコンでは、変数はプログラミングの土台になるもので非常に大切です。この節ではもう少し変数について説明を加えておきます。

変数のタイプ

前に、数値には単精度、倍精度などの型があると申しあげましたが、変数についても同じことがいえます。

たとえば変数Ａを整数で扱いたいのでしたら、うしろに％（パーセント）をつけてＡ％とします。また、単精度としてならＡ！、倍精度としてならＡ＃というようにそれぞれ！（イクスクラメーション）、＃（シャープ）をつけます。

A％
A！
A＃

これら３つの型をマンションにたとえるとすると、整数型変数は２ＤＫ、単精度型変数は４ＤＫ、倍精度型変数はぐっと広くなって８ＤＫということになります。

整数型変数
単精度型変数
倍精度型変数

そして、部屋の広さに応じて家具調度品が入っていますから、いざ引越しというときでも、２ＤＫの人は比較的速く整理をすますことができますが、８ＤＫとなるとずいぶん時間がかかります。

コンピュータについても同じことがいえ、処理時間は整数型がもっとも速く、単精度型、倍精度型と続きます。

ゲームなどのように、機敏な動きを必要とする場合には、変数は整数型で扱ったほうがずっと速くなります。

しかし、給与計算、請求書づくりなどでは、いくら速いといっても整数型、単精度型では不正確きわまりないですね。

もう１つの型宣言の方法

いま説明しましたように、ゲームでは単精度、倍精度の数値を扱うことはほとんどありませんから、変数はすべて整数扱いと見なしてもかまわないと思います。そんなときに、たくさんある変数にいちいち％をつけていてはキーインも大変です。これを解消する方法として、次の型指定文があります。

DEFINT

```
DEFINT A-Z
```

ＤＥＦＩＮＴ（デファイン・インテジャー）、つまり「整数定義をする」ということ、何を整数に定義するかといいますと、Ａ－Ｚ、Ａ

からZまでのアルファベットで始まる変数を、ということになります。この宣言文をプログラムの初めで設定しておけば、わざわざ%をつけなくとも、コンピュータがかってに判断して処理してくれます。

それでは、ほとんどの変数は整数型として扱いたいが、そのうちいくつかは単精度型、倍精度型として扱いたいというときはどうしたらよいでしょうか。これにも解決法があります。整数の場合と同じように、次の定義文を使います。

DEFSNG
DEFDBL

```
DEFSNG C
DEFDBL X
```

●整数扱いの変数がほとんどで、単精度、倍精度の変数が少ない場合、メモリの能率から考えると上の定義文を使わずに「!」「#」を使ったほうがよいでしょう。というのは、上の例で、変数Xだけを倍精度にしたつもりでも、X1、X2など整数扱いでよい変数があっても、すべて倍精度とみなされてしまうからです。

DEFSNG(デファイン・シングル)、DEFDBL(デファイン・ダブル)を使います。

上の例ですと、Cで始まる変数は単精度型として、Xで始まる変数は倍精度型として扱いなさい、ということになります。

すでに「DEFINT　A－Z」ですべての変数を整数扱いにする宣言がなされていても、そのあとで上の2つの宣言をすれば、自然にCおよびXで始まる変数は、それぞれ単精度、倍精度として扱われます。くりかえしますが、「DEFINT　A－Z」のあとで宣言しないとダメですよ。

34ページに文字型変数がでてきましたが、これについても同じことがいえます。あるきまった変数を文字型として扱いたければ、いちいち$マークをつけず、次のように宣言しておけばよいでしょう。

DEFSTR

```
DEFSTR P,Q,R
```

デファイン・ストリングと読みます。上の側では「P、Q、Rで始まる変数は文字型ですよ」ということになります。数値の場合にもこのように、宣言したい変数名をコンマでくぎって指定できます。

変数名の長さ

これまでは変数としてはX、Yなどのように1文字しか使いませんでしたが、最大40文字まで識別できます。

ですから、給与を扱う変数でしたらKではなくKYUYO、得点用の変数でしたらSCOREなど、わかりやすい表記をしておくとよいでしょう。ただ、あまり長いとかえって不便ですから念のため。

この節の内容は徐々にわかってくると思います。あまりむずかしく考えないでください。

2·6 画面をカラーに

ちょっと退屈な話が続きましたので、ここでは気分転換して画面をカラーにしてみましょう。

画面モードの設定

カラーに入る前に、画面モードについてご説明します。

いま、あなたの画面の文字は大きいですか、それとも小さいですか。おそらく大きくて見やすい文字だと思います。

画面はふつう電源を入れたとき横80桁、縦20行となっています。この桁数と行数は本体背面のディップスイッチによってハード的に変えることもできますが、モード指定のコマンドがありますので、これを利用することにしましょう。

〈注〉この節の記事は、1・3節のユーティリティプログラムを実行したままの画面モードに合わせてありますので、あらためてシステムを立ち上げた方にはつじつまが合わない点がありますがご諒承ください。

画面の桁数と行数を指定するには次のＷＩＤＴＨ（ウィズ）文を使います。

```
WIDTH A,B
```

（A：横の桁数、B：縦の行数）

A、Bはそれぞれパラメータで、それぞれ横の桁数、縦の行数をあらわします。そして、Aは80、40のいずれかを指定できますし、Bには25、20の２通りの行数指定ができます。

●N－BASICモードではパラメータAは80、40のほかに72、36も指定できます。

どれを使ったらよいという基準はありませんので、選択はあなたにおまかせします。

では次のようにキーインしてみてください。

```
WIDTH 80,20 ⏎
```

（80桁×20行にする）

いかがですか？　カーソルがずいぶん小さくなってしまったでしょう。

それでは、もとの画面にもどしましょう。次のようにします。

```
WIDTH 40,20 ⏎
```

もとの状態にもどりましたね。このように、いろいろなモードをためしてみてください。

次にもう１つ、画面構成のとき大切なコマンドがあります。カラー指定をしたり、ファンクションキー表示の有無を決めるＣＯＮＳＯＬ

CONSOLE

E（コンソール）文です。書式は次のとおりです。

```
CONSOLE A,B,C,D
```

- D：0ならモノクロ、1ならカラーモード
- C：0ならファンクションキーを無表示、1なら表示
- B：スクロール行数
- A：スクロール開始行

スクロール開始行

パラメータAはスクロール開始行です。画面を何行目から使うか、くらいに考えてください。画面の1番上は第0行となっていますので、ふつう、Aは0に指定します。

スクロール行数

パラメータBはスクロール行数です。これはWIDTH文で指定した行数に合わせておいてください。20行指定でしたら20、25行指定でしたら25です。

現在、画面の下にファンクションキーの表示がありますね。これを表示したり消したりするのがパラメータCです。0にすれば消して、1にすれば表示します。

モノクロモード
カラーモード

最後のパラメータDがカラー画面にかかわってきます。ここが0だとモノクロモード、1ならカラーモードになります。

文字をカラーで表示する

それでは実際にやってみましょう。次のようにキーインします。

```
WIDTH 40,20
CONSOLE 0,20,0,1
```

- 1：カラーモードにする
- 0：ファンクションキー表示を消す

画面はどうなりましたか？　「Ok」がでて、その下でカーソルがブリンクしていますね。カラーモードの指定をしましたが、文字はカラーになっていません。おかしいですね。

しかし、これでいいのです。カラーモードの指定をしたということは、画面をカラーにする準備ができたということです。ですから、あとは、どんな色にするか指示してやればカラー表示されるのです。

COLOR

カラーモードの色指定にはCOLOR（カラー）文を使います。

次のようにキーインしてください。すると画面は次のページのようになります。

```
COLOR 1 ⏎
```

- 1：青色のカラーコードです

```
Ok
COLOR 1
Ok
■
```

↑ Ok：青色で表示されました

「Ok」という文字が青色になりましたね。それでは次の場合はどうでしょうか。

```
COLOR 2 ↵
```

今度は赤色になりました。この状態でいろいろな文字を画面に書いてみてください。すべて赤色で表示されますね。

このように「COLOR」のうしろの数字を変えると、いろいろな色が楽しめます。この数字のことを**カラーコード**といい、右の表のように、それぞれの色と対応しています。

カラーコード

カラーコード	カラー
0	黒
1	青
2	赤
3	紫
4	緑
5	水色
6	黄
7	白

カラーコード0は黒ですから、これを指定すると画面の色と同じになり、何も見えなくなります。そんなときは、見えないながらも [C] [O] [L] [O] [R] [スペース] [7] [↵] とキーインすれば、白色が指定されますから、もとの状態にもどります。

各自ためしてみてください。

画面をクリアする

いままで私たちは画面をクリアする場合、ホームクリアキーを使っていました。これはワンタッチでできますから簡単でいいのですが、これから学ぶプログラム中では、そうはいきません。

ちょうど本節は画面にかかわる内容ですので、クリアする命令も覚えてしまいましょう。

CLS

これには**CLS（クリアスクリーン）文**を使います。次のようにキーインしてください。

```
CLS ↵
```

どうですか？　いままで画面にかいた文字が、またたくまにクリアされましたね。

以上でこの章は終りです。簡単な演算から画面構成まで、基本的な知識として欠かせないことを述べてきました。さて、いよいよ次章からはプログラミングです。

第3章

簡単な プログラム

3・1 まず、小さなプログラムから

ダイレクトモードとプログラムモード

私たちは第2章でキーボードからいろいろな命令をキーインしました。これはコンピュータに「直接」、つまり「ダイレクト」に問いかけていたわけですから、答もすぐに返ってきました。

すぐに解答が得られるので、せわしい現代では大変便利に思えますが、そのつど質問内容をキーインしなければなりません。

このように直接コンピュータに質問し、すぐに答が得られる形式を**ダイレクトモード**といいます。

ダイレクトモード

これに対して、すぐに答は得られませんが、一度キーインしてしまえば、類似した質問を何回もできる、という便利な形もあります。これが、これから勉強する**プログラムモード**です。

プログラムモード

いまはわからなくても、少しずつ具体例を実践していくにつれて、身についていくと思います。

行番号をつける

30、31ページをごらんください。

私たちはダイレクトモードで次のような操作をしました。

```
X=5 ⏎     ← Xに5を代入
Y=3 ⏎     ← Yに3を代入
? X+Y ⏎   ← XとYの内容の和を表示しなさい
```

●前章の続きでカラーモードのままの人はモノクロモードにしてファンクションキーを表示させておいてください。

これは2つの数をたした結果を画面に表示させるものでした。そして、この表現方法をダイレクトモードというのですね。

それでは、このダイレクトモードをプログラムモードに変えてみましょう。ホームクリアキーで画面をクリアしたのち、何もいわずに次のようにキーインしてください。

[1] (スペース) [X] [=] [5] ⏎

[2] (スペース) [Y] [=] [3] ⏎

[3] (スペース) [?] (スペース) [X] [+] [Y] ⏎

次のような画面になりましたか？



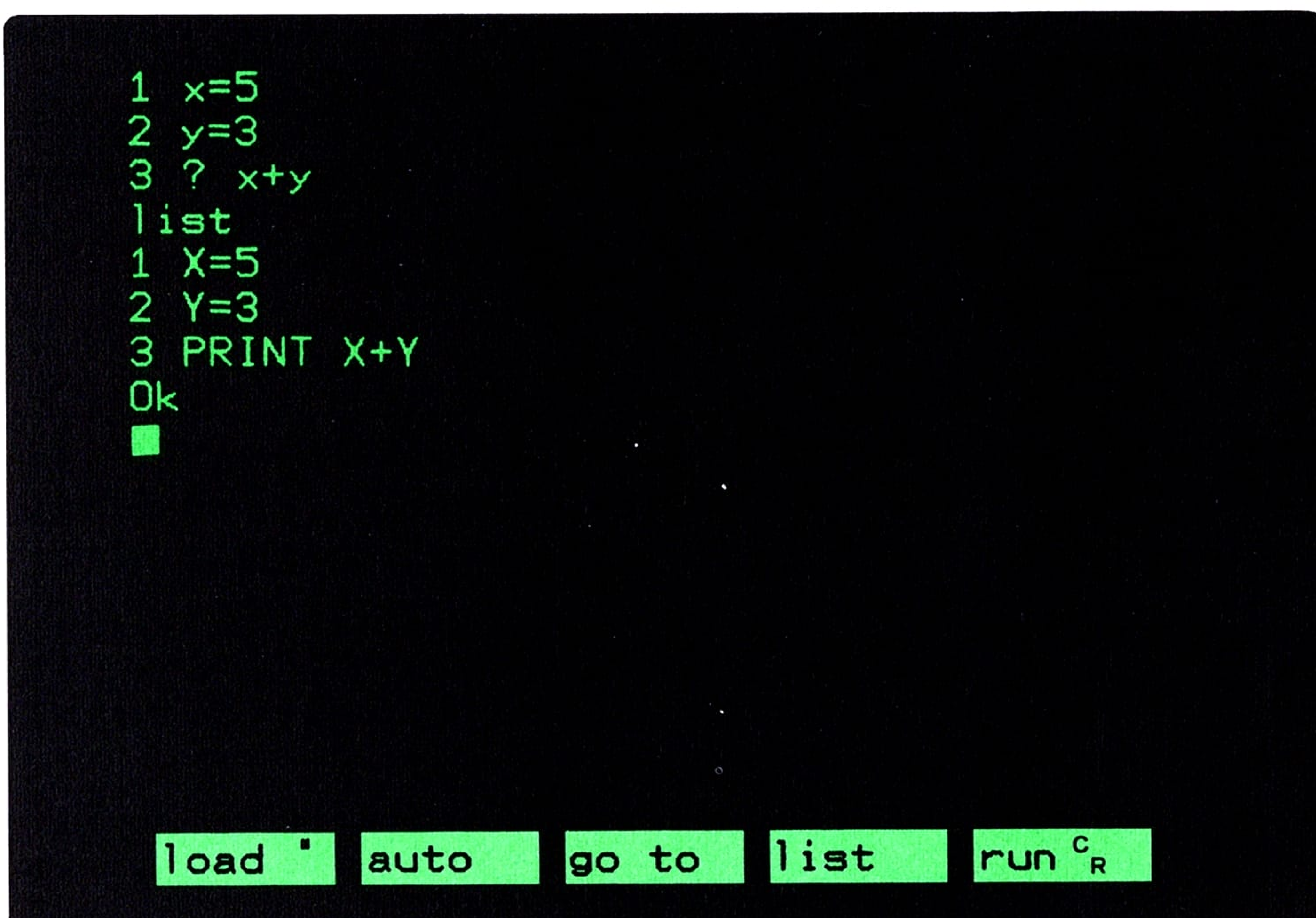
この画面では小文字でキー入力しましたが、大文字でもかまいませんよ。

画面には「Ok」も表示されないし、「3」の下でカーソルがブリンクしているだけですね。どうしたのでしょうか？　何が起こったのでしょうか？　答なんてぜんぜん表示されませんね。

しかしこれでいいのです。これがりっぱなプログラムなのです。

プログラム

この状態で、キーボードの上にある [f・4] キーを押してから [↵] キーを押してみてください。画面は次のようになりました。



[f・4] キーを押したとき画面に「list」と表示されました。これは、現在コンピュータに記憶されている「プログラムのリスト」を表示させるものです。「list」は [f・4] キーを使わずに直接 [L] [I] [S] [T] とキーインしても同じです。ただ、[f・4] キーを使ったほうが1回の操作でできますから楽ですね。

●上の画面のファンクションキー表示の右から2番目つまりf・4キーにlistが定義されているのがわかりますね。

プログラムリスト

いいですか？「list」と表示して次に ⏎ キーを押すと、この命令が実行されて、プログラムリストが画面に表示されるのです。

「list」の下に３行分表示されているのがプログラムリストです。

●プログラムモードでは、「？」と入力しても、あとでリストをとると「ＰＲＩＮＴ」に変換されます。これで「？」と「ＰＲＩＮＴ」が同じように使えることがわかりましたね。

気をつけてみますと、小文字でキーインしたのが、大文字に変わっていますし、「？」は「ＰＲＩＮＴ」になってしまいました。しかし内容は、あなたが入力した上の３行と同じなのです。

このように、入力は小文字でもコンピュータは大文字に変換して表示します（ただし、ダブルクォーテーションマークでくくったものはこのかぎりではありませんが、これはあとでわかると思います）。

これであなたがキー入力したプログラムがコンピュータに記憶されました。そしてそのプログラムとは、「変数Ｘに５を、変数Ｙに３を入れ、その和を教えてください」というものです。**あなたがコンピュータにやらせたいこと、それがそのままプログラム**なのです。ただ、コンピュータにはあなたのしゃべる言葉が通じないから、**コンピュータにわかる形***にしてあげなければなりません。それが、いま画面に表示した３行のプログラムなのです。

＊ベーシック、ベーシックとよくいわれますが、これがそのＢＡＳＩＣです。実際にはＢＡＳＩＣは同時通訳者みたいなもので、これをさらにマシン語に変換しています。

その３行だけをもう一度次に示します。

```
1 X=5
2 Y=3
3 PRINT X+Y
```

プログラムです

行番号といいます

行番号

一番左側の列の１、２、３という数字は行番号といいます。コンピュータは、この行番号をたよりにあなたが入力したプログラムを実行していきます。ですから、たとえてみれば、一種の標識みたいなものですね。

行番号は別に１、２、３というように１きざみでなくてもかまいません。これはあとでわかることですが、むしろ余裕をもたせたほうがよいのです。ふつうは10きざみくらいにしています。

画面をクリアすると……

ダイレクトモードでは画面をクリアするとステートメントはあとかたもなく消えてしまいました。

ところがプログラムモードでは消えません。ちゃんとコンピュータの記憶領域（メモリ）に残っているのです。ですから、先ほどのように**ＬＩＳＴコマンド**を実行すれば、ふたたび画面に表示してくれます。

LISTコマンド

さっそくためしてみましょう。

まずホームクリアキーで画面を消去します。次に f・4 キーを押してリターンです。

```
list
1 X=5
2 Y=3
3 PRINT X+Y
Ok
■
```

いかがですか？　プログラムリストが表示されましたね。

プログラム入力上の注意

これで「プログラムとはこんなものか」となんとなくご理解いただけたと思いますが、もう一度復習しますと、プログラムは

〈行番号〉〈ステートメント〉

から成り立っています。

そして、「行番号」と「ステートメント」の間には

スペース

を入れましょう。カーソル移動キー → で1スペースあけるのではありませんよ。スペースキーを1度たたくのです。

もっとも、スペースをあけなくてもコンピュータはかってにスペースをとってくれますが、スペースを入れるクセをつけておいたほうがよいと思うからです。と申しますのは、たくさんある命令の中にはスペースを入れないとエラーになってしまうものが多いからです。スペース挿入を習慣づけておけば、プログラムもずっと見やすいものになります。

そして一番大切なこと、1行キーインし終ったら、必ず

リターンキー（⏎）を押す

ことです。最初のうちは、この操作を忘れる人が多いようですが、プログラムが思いどおりに実行されません。

3・2 プログラムの実行

さて、プログラムができあがりましたので、次のステップへ進みましょう。それは、私たちがいま望んでいる「答を画面に表示させる」ことです。このことを「プログラムを実行する」といいます。

プログラムの実行

RUNコマンド

プログラムを実行させるには**RUN（ラン）コマンド**を使います。

R　U　N　⏎

このように1字1字キーボードから入力してもかまいませんが、ファンクションキー表示の一番右側を見てください。

run $^{C}{}_{R}$

という文字があります。つまり f・5 キーにこれが定義されているのです。

キャリッジリターン

「CR」は「キャリッジリターン」と読み、⏎ キーを押すのと同じ機能をもっています。いいかえれば、f・5 キーを押せば、R　U　N　⏎ とキーインしたのと同じことになるのです。

ではさっそく実行してみましょう。 f・5 キーを押してください。

```
list
1 X=5
2 Y=3
3 PRINT X+Y
Ok
run ←―――――――― プログラムの実行
 8 ←――――――――― 答
Ok
█
```

あっというまに「8」という答が表示されました。

これは何回でもくりかえすことができます。この状態で f・5 キーを押せば、ふたたび同じ答が得られますし、ホームクリアキーで画面をクリアして f・5 キーを押しても、やはり同じ結果が得られます。

3・3 プログラムのセーブとロード

さあ、小さなプログラムを作ることができました。

初めてのパソコンの勉強で少々つかれたのではありませんか？ そんなとき、「ああ、肩こりもしてきた、また明日やろう」と思う人もいるかもしれません。そうなると、どうしても電源を切らなければなりません。せっかく作ったプログラムはどうなるのでしょうか。電源を切ってもメモリに保存されているのでしょうか？ 一度シミュレーションを行ってみましょう。

電源を切る

ディスクを使用している人はまずディスケットを取り出してください。そして、次の順序で電源ＯＦＦにします。

❶ **PC-8801本体の電源OFF**
❷ **ディスプレイの電源OFF**

これで、きょうの作業はすべて終りました。

そして翌日。きのうの続きをやるためにコンピュータの前にすわり、例によって電源を入れます（ディスクを使用する方もそうでない方も1・1節を見ながら電源ＯＮにしてください）。

「さて、きのうのプログラムを画面にだしてみよう」と思い、前日覚えた要領で画面をクリアしてから　f・4 ⏎ としました。

```
list
Ok
■
```

ところが、画面は上のようになり、何もでてきません。

そうです。前日のプログラムは消えてしまったのです。もう一度キーインしなければならないのです。

短いプログラムならいいですが、長いものになると泣くに泣けませんね。そこで、せっかく作ったプログラムをなんとか保存しておく方法はないか、ということになるわけです。

もう一度、42ページの方法でプログラムを入力してください。そして、ＬＩＳＴ命令で間違いがないかどうか確認してください。

カセットテープに保存する

この項はカセットテープの方が対象ですが、ディスク使用の方もカセットを使う機会もあると思いますので、お読みください。

SAVEコマンド

カセットテープとかディスクにプログラムを保存することを「**セーブする**」といい、ＳＡＶＥコマンドを使います。そして、どんなプログラム名で保存しておくのか、プログラムに名前をつけなければなりません。ここでは仮りに「prog」としておきましょう。

次のように入力し、⏎キーはまだ押さないでください。

```
SAVE "CAS1:prog" ■
```

プログラムをカセットにセーブ
カーソルはまだここにある
プログラム名（小文字にしてください）

●「save"」はｆ・６キーに定義されていましたね。ｆ・６キーを押すと簡単です。

「ＳＡＶＥ」も「ＣＡＳ」も小文字でもかまいません。

カセットの電源は入れてありますか？　準備できましたら、ボリュームを中くらいより大きめ、音質のつまみも中くらいより少し高めに調整してください。

次に録音状態にしてから ⏎ キーを押します。

するとカセットへのセーブ処理が開始し、少したって終わると、画面に「Ok」が表示されます。

```
SAVE "CAS1:prog"
Ok
■
```

セーブ完了です。ここでカセットをとめてください。

もし、ピーッと鳴ってエラーメッセージが表示された場合には、セーブ失敗ですから、ボリューム、音質を調整してもう一度やりなおしてください。

さて、これでセーブできたわけですが、確認しておく必要があります。せっかくセーブしたと思ったのに、あとでカセットテープから取り出す（ロードするといいます）ことができなかったということがけっこうあります。

まず、カセットテープを巻き戻しておいてください。そして、次のようにキーインします。

```
LOAD? "CAS1:prog" ⏎
```

↵キーを押してしまってもかまいませんが、「ＬＯＡＤ」のうしろの**「？」を入れ忘れていませんか？**　確かめてください。

だいじょうぶでしたら、カセットテープを再生状態にします。するとテープが回り始めて、もし　「prog」が見つかると画面の右上に「＊」（アスタリスク）が現れ点滅します。このアスタリスクが点滅しているときにコンピュータは、テープのプログラムと本体のメモリ内にあるプログラムが合っているか照合しているのです。そして間違いがないと、「Ok」が表示されて終りです。

〈注〉ここでのセーブ、ロードは「ＣＡＳ１：」を使いましたが「ＣＡＳ２：」でも同じです。ただしセーブしたときの形でロードしなければなりません。標準添付のデモテープは後者を使っています。

この、確認作業のことを**「ベリファイ」**といいます。ベリファイ開始から終了までの画面表示は次のようになります。

ベリファイ

```
LOAD? "CAS1:prog"                                   *
Found:prog
Ok
■
```

アスタリスク

プログラムのロード

カセットテープにセーブしたプログラムを必要なときにメモリ上に取り出すことを「プログラムをロードする」といいます。

ＬＯＡＤコマンドを使います。

いま、ベリファイをしたときもＬＯＡＤを使いましたが、そのときはうしろに「？」をつけていました。今度は「？」をとって、次のようにキーインします。

LOADコマンド

```
LOAD "CAS1:prog" ↵
```

必ず小文字にする

プログラム名は小文字でセーブしましたから、大文字ではいけません。もし大文字でキーインすると、次のような表示がでて飛ばしてしまいます。

●「ｌｏａｄ」はｆ・１キーに定義されていますね。

```
LOAD "CAS1:PROG"
Skip:prog
```

「ｐｒｏｇ」を飛びこえてしまいました

間違えたらもう一度初めからやりなおしてください。ロードが成功したときの画面表示はベリファイの場合と同じです。

ディスケットにセーブする

この項では、私たちの作ったプログラムをディスケット（「ディスク」と同意）に保存する方法を学びます。

ディスケットにプログラムをたくわえておくことを**「セーブする」**といいます。英語で「セーブ・マネー」というと貯金のことですね。なんとなくわかるような気がしませんか？

このセーブする作業もプログラミングにとって大切な仕事の１つです。それではとりかかりましょうか。

ディスクは起動していますね？　そして、ディスケットはドライブ１に入っていますか？　確かめてください。

SAVE

ディスケットにプログラムをセーブするときもテープの場合と同じようにＳＡＶＥコマンドを使います。

プログラム名はやはりテープのときと同じ「prog」を使います。次のようにキーインしてください。

```
SAVE "1:prog"
```

（prog ― プログラム名、1 ― ドライブ番号）

「１：」はドライブ番号を示しています。もしドライブ２のディスケットにセーブしたいのでしたら「２：」とします。いまはドライブ２にディスケットが入っていませんね。また、ドライブ１の場合「１：」は省略できます。

●プログラム名は大文字でも小文字でもかまいませんが、ＳＡＶＥしたときの形でＬＯＡＤするのが原則です。

「ＳＡＶＥ」は大文字でも小文字でもかまいませんが、「prog」は小文字にしてください。

確認して間違いありませんでしたら ↵ キーを押します。間もなく「Ok」と表示がでてセーブ終了です。

```
SAVE "1:prog"
Ok
█
```

セーブされたかどうかの確認は次のようにキーインします。

```
FILES ↵
```

画面は次ページのようになり、間違いなくセーブされていました。

```
save '1:prog'
Ok
files
demo   .n88 7  tutor .n88 12 dskut1.n88 4  dskut2.n88 9  ngen  .n88 13
@load .n88 1  @exst *    2  demoex.n88 3  *     *    1  *demo0*    4
*demo1*     1  *demo2*     1  *tutor*     1  $pict0 dat 4  $pict1 dat 4
$pict2 dat 6  $pict3 dat 6  $pict4 dat 10 $pict5 dat 12 $pict6 dat 12
$pict7 dat 14 $pict8 dat 14 prog   .     1
Ok
■
```

（prog → セーブされている）

ディスクからロードする

ディスケットにセーブされたプログラムをメモリにロードするには LOADコマンドを使います。書式は次の通りです。

LOAD

```
LOAD "1:prog"
```

（1 → ドライブ番号）

セーブの場合と形が似ていますね。

ドライブ1のディスクからロードする場合、「1：」は省略してかまいません。⏎キーを押すと「Ok」が表示されてロード成功です。

```
LOAD "1:prog"
Ok
■
```

もし、プログラム名を大文字でキーインすると、ピーッというブザー音がしてエラーメッセージが返ってきます。

```
LOAD "1:PROG"
File not found
Ok
■
```

そんなときは、もう一度やりなおしてください。

セーブ、ロードの方法がこれでマスターできましたので、これからはプログラミングの学習が終ったら必ず、そのとき作ったプログラムをセーブするようにしましょう。なお、プログラムを修正して同じプログラム名でセーブすれば、古いプログラムは消えて新しいものに入れかわります。

3・4　任意の数のたし算

私たちは3・1節で小さなプログラムを作りました。しかし、これはダイレクトモードと違って何回でも実行できますが、毎回「5＋3」ばかりです。これでは実用的ではありませんね。

そこで、今度は、私たちが指定する任意の数についてコンピュータに計算させるようにしてみましょう。

それには、変数ＸとＹに５、３という定数を入れておかないで、プログラムの実行のたびに好きな数を指定できるようにすればよいわけです。

この条件を満たしてくれる便利なコマンドがあります。

INPUT

それは**ＩＮＰＵＴ文**です。

みなさんの前のディスプレイには、先ほどの小さなプログラムが表示されていますか？　まだでしたら、カセットテープあるいはディスクからロードして、ＬＩＳＴコマンドでリストを画面に出してください。もう、やり方はわかりますね。

ステートメントを変更する

いま画面に「prog」という名前のプログラムリストが表示されています。行番号１と行番号２で変数Ｘ、Ｙの内容を指定しているわけですから、ここを、任意の数を指定できるように変更すればよいのです。次のリストをキーインしてください。

```
1 INPUT X
2 INPUT Y
```

スペースを１つ入れる（行番号１：「1」と「INPUT」の間、「INPUT」と「X」の間）

スペースを１つ入れる（行番号２：「2」と「INPUT」の間、「INPUT」と「Y」の間）

できましたか？　スペースは必ず入れるようにしましょう。ＩとかＰとかは目に見える１字分、スペースは目に見えませんが、空白という形で、やはり１字分とっているのです。

それから１行キーインし終ったら、必ずリターンキーを押してください。そうしませんと、ステートメントをコンピュータが記憶してくれません。また、最初のうちよくあるのですが、行番号１でリターンキーを押さず行番号２で押すと行番号２のステートメントは行番号１に含まれてしまいます。

画面は次のようになっています。

```
list
1 X=5
2 Y=3
3 PRINT X+Y
Ok
1 INPUT X
2 INPUT Y
■
```

上半分には、私たちが最初に作ったプログラムリスト、下には、いま作った2行が表示されています。コンピュータはこの2つを別々に記憶しているのでしょうか？

ホームクリアキーで画面をクリアしてください。そしてリストコマンドでプログラムリストを表示させてみましょう。

```
list
1 INPUT X
2 INPUT Y
3 PRINT X+Y
Ok
■
```

おや、最初に入力したステートメントは消えてしまいましたね。

このように、コンピュータは、同一の行番号の場合、あとから入力したステートメントを優先して記憶します。

プログラムの実行

さて、それではプログラムを実行させてみましょう。(f・5) キーを押せばよかったですね。

RUNコマンドによりプログラムの処理が始まり、画面には次のようにクエッションマークが表示されて、そのわきでカーソルが点滅しています。

```
run
? ■
```

これはどういうことかといいますと、コンピュータがあなたに「何か入力してください」と問いかけているのです。

あなたの好きな数をキーインしてみてください。数を入力したら必ずリターンキーを押します。こうしませんと、コンピュータはいっこうに処理をすすめてくれません。

たとえば [1] [0] [0] [↵] としますと、画面は次のようになります。

```
? 100
? ■  ←―――――――――― カーソルがブリンク
↑
└――――――――――――――― クエッションマーク
```

先ほどと同じように、「？」のうしろでカーソルがブリンクしています。あなたに「2つ目の数を入力してください」とコンピュータが問いかけています。[2] [0] [0] [↵] とキーインしてみましょう。

```
? 100
? 200
 300
Ok
■
```

正解が表示されましたね。プログラムが正しく実行されたようです。

3・2節では変数X、Yにそれぞれ5と3が入っていましたが、今度は何が入っているでしょうか？　ダイレクトモードで調べてみましょう。やり方はわかりますね。

```
? X  ←―――――――――― Xの内容を表示しなさい
 100
Ok
? Y  ←―――――――――― Yの内容を表示しなさい
 200
Ok
■
```

上のようになりましたか？

変数XとYには、いま私たちが入力した数が入っています。

それでは、もう一度 (f・5) キーを押してプログラムを実行してみましょう。今度は、最初は300、次に400を入力してみてください。そして同じようにして、変数XとYの内容を調べてみてください。

いかがですか？　前の変数XとYの内容100と200が追い出されて、今度は新しく入力した300と400が入っていましたね。

このように変数の中味はいつも変えることができるのです。

プログラムのけじめ

この節で作ったプログラムをRUNコマンドで実行させますと、1回だけ処理をして終りになります。しかし、プログラムの中には終りを示すステートメントは入っていません。これでもいっこうにさしつかえないのですが、けじめをつけるためにエンドマークを入れておきましょう。**ENDコマンド**を使います。 END

次のように、行番号を4としてステートメントをキーインします。

```
4 END ← プログラムの終り
```

これで礼儀作法にあった形になりました。もう一度画面をクリアして実行してみてください。

もう少し続けたい

プログラムが完成したものの、現在は、2回以上実行したければ、そのつどRUNコマンドを使わなければなりません。これではちょっと、という感じですね。

そこで、だまっていても処理をくりかえしてくれるようにしてみましょう。そのためには、いま入力したENDコマンドに行く前に、行番号1へ処理を戻してしまえばよいわけです。こんなとき使う命令に**「GOTO」(ゴートゥ)**があります。 GOTO

行番号3と行番号4の間にこのコマンドを入れるわけですが、行番号は整数でなければいけませんから、もう入れる余地がありません。そこで、行番号を1きざみではなく10きざみくらいにして、もう少しゆとりをもたせましょう。

といっても、各ステートメントを初めからキーインする必要はありませんよ。いまコンピュータに記憶されているプログラムの行番号をつけかえてしまえばよいわけです。この目的を達成してくれるのが、**「RENUM(リナンバー)」コマンド**です。「RENUMBER」からきているのだと思われますが、「行番号をつけかえる」ということです。 RENUM

次のようにダイレクトモードでキーインしてください。

```
RENUM ⏎
```

できましたら、

```
(f・4) ⏎
```

として、リストを画面に表示させてみてください。次のようになりま

したか？

```
RENUM
Ok
list
10 INPUT X
20 INPUT Y
30 PRINT X+Y
40 END
Ok
■
```

ずいぶん便利ですね。いままで１、２、３、４と１きざみだった行番号が10きざみに変身してしまいました。

ここでＲＥＮＵＭについてもう少し補足しておきましょう。

ＲＥＮＵＭの書式は次の通りです。

```
RENUM A,B,C
```

これは「行番号Ｂを行番Ａに変えて、以降をＣの間隔でつけかえなさい」という意味です。

先ほどのＲＥＮＵＭを実行する前のプログラムをこの書式に準じて実行するとしますと、正式には次のようになります。

```
RENUM 10,1,10 ↵
```

つまり、「行番号１を行番号10にして、10きざみにつけかえる」ということです。

しかし、私たちはこのような表記はしませんでした。ＲＥＮＵＭのうしろの数字はすべて省略してしまいました。

そうです。パラメータを省略すると、コンピュータは自動的に、プログラムの先頭行を10として、それ以降を10きざみにつけかえてくれます。

ＲＥＮＵＭコマンドはプログラミングの過程で大変よく使います。私たちも少しずつ実践して覚えていくようにしましょう。

ＧＯＴＯ

●「ＧＯＴＯ」はファンクションキーｆ・３にも登録されていますから、ｆ・３を押せば１回でキーインできます。

さて、行番号に余裕ができましたので、私たちのいまの課題「ＧＯＴＯステートメントを作ること」をすませてしまいましょう。行番号を35として次のようにキーインします。

```
35 GOTO 10 ↵
```

行番号を35としたからといって、別に、行番号30と40の間にインサートするというわけではありませんよ。いまのようにキーインすれば各ステートメントはちゃんと行番号順になってしまいます。画面をクリアしてリストを見てみましょう。

```
list
10 INPUT X
20 INPUT Y
30 PRINT X+Y
35 GOTO 10
40 END
Ok
■
```

どうですか？　うまくいっていますね。

さあ、それではＲＵＮコマンドで実行してみてください。

今度は１回きりではなく、何回でも実行できますね。しかし、何回でも実行できるのはいいのですが、いつまでたっても終りになりません。どうしてでしょうか？

それは、上のプログラムをごらんください。コンピュータは行番号10、20、30の順で処理を実行し、行番号35のＧＯＴＯ文でふたたび行番号10へ戻ってしまいます。つまり、10→20→30→35→10→20というように行番号10と行番号35の間をぐるぐる回っているからです。この、ぐるぐる回ることをループといい、上のプログラムのように終りのないループを**エンドレスループ**といいます。

ループ
エンドレスループ

エンドレスループでは、処理の実行を止めるには STOP キーを押すしかありません。STOP キーを押してみてください。

```
?
Break in 10
Ok
■
```

行番号10でブレークがかかった、つまり処理を中断したということです。いちいちこのように止めるのでは非実用的です。次の節では、もう少しくふうしてみましょう。

3・5　条件をつける

「条件をつける」とは、私たちの入力した値についてコンピュータが、ある一定の条件判断をしてくれることをいいます。そして、その条件を満たしていれば、私たちがあらかじめコンピュータに記憶させておいた処理をしてくれます。

具体的にいいますと、たとえば前ページ上のプログラムで、行番号10で変数Xの入力指示があったとき、もう終りにしたいと思ったら0と入力してやると、このプログラムを終りにすることができるのです。つまり、「もしXが0だったならば、行番号40へ行きなさい」という条件文を設定しておくのです。

条件文の書式には次の3つがあります。

```
IF 〈条件〉 THEN GOTO 〈行番号〉
IF 〈条件〉 THEN 〈行番号〉
IF 〈条件〉 GOTO 〈行番号〉
```

IF〜THEN文
IF〜GOTO文

上の2つを「IF〜THEN文」、3つ目を「IF〜GOTO文」といいます。どれも同じ機能をもっていますから、みなさんの好きな形をお使いください。

また、IF〜THEN文では必ずしもGOTO文で飛び先を指定するばかりではありませんが、それはあとで出てきたときに説明することにします。

私たちのプログラムでは、変数Xの値を入力し、それが0かどうかをコンピュータに判断させ、0なら終りにしてしまえばよいわけですから、行番号を15として次のようにキーインします。

```
15 IF X=0 THEN GOTO 40
```

● X＝0という条件を満たしていなければ、処理は次の行番号20に移ります。

この本では、もっともスタンダードなIF〜THEN文を使うことにします。キーイン後リターンキーを押し忘れないでください。

それでは実行してみましょう。何回か演算をさせてみてから、Xに0を入力してみてください。次のようになります。

```
 8
? 0
Ok
■
```

3・6 プロンプト文を入れる

前節まででー応プログラムらしくなり、２数のたし算を何回でもできるようになりました。しかし、コンピュータから私たちへの問いかけは、画面にただ「？」が表示されるだけです。私たちには何を入力すればよいかわかっていますからこれでよいのですが、もしほかの人がこのプログラムを実行するとしたら、「何だろう」と頭をかしげてしまいますね。

そこで、コンピュータの問いかけが何であるか誰にもわかるように、**プロンプト文**（メッセージ）を入れることにしましょう。

処理内容を変更する

私たちが作ったプログラムの内容は「２つの数の和を計算する」ものでした。しかしこれではあまり実用性がないので、「納品書づくり」を前提にしたプログラムに変更することにします。計算式は、

単価×数量＝金額

となりますから、単価に変数Ｘを、数量に変数Ｙを使うとすれば

X×Y

を計算すれば、答としての「金額」を得ることができます。プログラムリストを画面に表示させてください。次のようになっていますね。

```
10 INPUT X
15 IF X=0 THEN GOTO 40
20 INPUT Y
30 PRINT X+Y
35 GOTO 10
40 END
```

このプログラムをいま説明した内容に変更するにはどうしたらよいでしょうか？　すぐわかりますね？　そうです、行番30の「Ｘ＋Ｙ」の「＋」をかけ算記号「＊」にすればよいのです。

やってみましょう。

カーソル移動キーでカーソルを「＋」のところに重ねます。そして［＊］キーを押し、次にリターンキーを押せばよいのです。必ずリターンキーを押さなければ変更したことになりませんよ。

ほんとうに変わったかどうか確かめてみましょう。画面をクリアしてから、次のようにキーインしてください。

```
list 30 ⏎
```


画面は次のようになります。

```
list 30
30 PRINT X*Y
Ok
■
```

うまくいったようですね。これで、「納品書づくりのプログラム」に変更できました。プログラムの名前は「ＮＯＨＩＮ」としておきましょう。

ちょっと横道にそれますが、この辺で今日のプログラムの勉強を終りにしたいという人は、**必ずこのプログラムをカセットテープかディスケットにセーブしておきましょう。**セーブ方法は3・3節をごらんください。

このような短いプログラムでしたらいいのですが、長いプログラムのセーブをおこたると、もう一度初めからキーインしなければなりませんので、うんざりしてしまいます。

プロンプト文の入れ方

行番号10のＩＮＰＵＴ文にメッセージを入れる場合を例にとります。

まず、ＬＩＳＴコマンドで10行を画面に表示してください。次に、カーソル移動キーでカーソルをＸに重ねてください。画面は次のようになっています。

```
list 10
10 INPUT X
Ok
```


そうしましたら、SHIFT キーを押しながら INS キーを１回押しインサートモードにします。そして次のようにキーインします。

" タ ン カ " ;

画面は次のようになっていますか？

```
list 10
10 INPUT "タンカ";X
Ok
```

（;）←セミコロンです

間違いなかったら、ここでリターンキーを押しましょう。

これで変数Xに対するメッセージが入りました。

ここで (f・5) キーを押してプログラムを実行してみてください。

```
run
タンカ? ■
```

今度はわかりやすくなりましたね。(STOP) キーを押してプログラムの実行を止め、行番号20についてもメッセージを挿入してみてください。手順はまったく同じです。メッセージは「スウリョウ」とします。

```
20 INPUT "スウリョウ";Y
```

さあ、完成しました。RUNコマンドで実行してみてください。

```
run
タンカ? 100
スウリョウ? 50
 5000
タンカ? 200
スウリョウ? 300
 60000
タンカ? ■
```

どうですか？　ずいぶんわかりやすくなったでしょう。

プログラムの実行を止めたければ、「タンカ」を聞いてきたときに0を入力すればよいのです。

●現在変数は電源を入れたときのままですからX、Yとも単精度型です。従って、その積が7桁以上になると7桁目が四捨五入されて次のような指数表現になります。

1.23456E＋06

E＋06は10の6乗です。

3・7　画面のモードを決めておこう

CONSOLE
WIDTH

いまあなたはどんな画面モードを使っていますか？　2・6節を思い出してください。すでに学習ずみですね。そこで私たちが学んだのはCONSOLE文とWIDTH文でしたね。

この章では、モノクロモードを使い、ファンクションキーは目ざわりですので表示しないことにします。また、画面の桁数と行数ですが、目の健康を考えて大きい文字の40桁、行数は、あとで作表をしますので25行にしておきます。

●作表の場合、縦20行のモードにしておきますと、タテのケイにすきまがあいてしまいます。

このモード設定はプログラムの先頭に入れておきます。次のようにキーインすればよいと思いますが、いかがでしょうか？

```
1 CONSOLE 0,25,0,0
2 WIDTH 40,25
```

モノクロモード
40桁×25行にする

これでもいっこうにかまいませんが、この２つのステートメントは１つにすることができます。

32ページをごらんください。コロン「：」を使って２つ以上のステートメントにしたものをマルチステートメントというのでした。

マルチステートメント

さっそく応用してみましょう。次のようにキーインします。

```
1 CONSOLE 0,25,0,0:WIDTH 40,25 ⏎
```

さあ、実行してみてください。少しは見やすくなりましたか？　しかし、もう少しくふうしておくとさらにコンピュータらしくなってきます。次の項で説明しましょう。

●マルチステートメントを使って２行を１行にしてしまいましたので、行番号２は不要となります。次のようにキーインしてください。

2 ⏎

これで行番号２のステートメントは消えてしまいました。このように、ある行を削除したいときは、行番号をキーインしてリターンキーを押せばよいのです。

画面のクリア

このプログラムを何回か実行したみなさんは、もうおわかりのことと思いますが、たとえば９回、10回と処理を続けた場合、それまでの表示がそのまま画面に残っていて上にどんどんスクロールしていきます。つまり、現在の入力メッセージが画面の一番下に表示されてしまいます。

これではちょっと見づらいと思いませんか？

そこで、１回の処理が終るごとに画面をクリアして、次の入力メッセージも一番上に表示されるようにしたらどうでしょうか？

そのためには、行番号10の処理に飛んでくる前に画面をクリアする必要があります。行番号30で「金額」を表示してから画面をクリアし、次いで行番号35のＧＯＴＯ文によって10行へジャンプするようにすればよいですね。行番号34として次の**ＣＬＳステートメント**を追加しましょう。

ＣＬＳ

```
34 CLS
```

リターンキーは押し忘れていませんか？　キーインできましたら、(f・5) キーで実行してみてください。

さあ、ＲＵＮさせてみて何か気づいたことはありませんか？

確かに画面はクリアされて次の処理のメッセージがでるのですが、私たちが知りたいと思っている「金額」の表示が果たしてあったのかなかったのか、ほとんどわかりませんね。

コンピュータの処理速度は非常に速いものです。行番号30のＰＲＩＮＴ文によって表示はされているのですが、すぐ画面をクリアして、行番号35のＧＯＴＯ文によって10行へジャンプしてしまうからです。

INKEY$関数

これを防ぐのによい方法があります。**ＩＮＫＥＹ＄（インキーダラー）**を使うのです。次のようにキーインして実行してみてください。

ＩＮＫＥＹ＄

```
32 Y$=INKEY$
33 IF Y$="" THEN GOTO 32
```

ＩＮＫＥＹ＄はキー入力のあった文字を与えるものです。行番号32では、何かキー入力があればその文字がＹ＄の内容になり、キー入力がなければ長さのない文字（**ヌルストリング**といいます。何もないということです）がＹ＄の内容になります。そして33行へ行きますが、ここでは条件文を使っていますね。ダブルクォーテーションの間に何も（スペースさえも）入っていません。つまり、Ｙ＄の内容がヌルストリングだったら行番号32へ行きなさいということです。キー入力があればＹ＄に当然なんらかの文字が入りますから、次の処理（この場合、行番号35）へ実行が移されます。つまり、**何もキー入力がなければ、32行と33行の間をグルグル回っている**わけです。

ヌルストリング

●ここでＩＮＫＥＹ＄ではなくタイマー（時間待ち）ステートメントを入れてもよいのですが、これについては後で説明しますから、そのときになってもう一度ここへ戻ってみなさんが工夫してみてください。

それでは実行してみましょう。数量の入力をすると金額が表示されて、そのままの状態が維持されます。次の処理へ移りたかったら、どのキーでもいいですから押してください。画面がクリアされて、次の入力メッセージが表示されます。

3・8　演算は別のステートメントで

はじめ3行からスタートした私たちのプログラムも、いろいろ条件をつけ加えていくうちに、いつの間にか10行になってしまいました。ＬＩＳＴコマンドでプログラムリストを画面に表示してみましょう。

```
1 CONSOLE 0,25,0,0:WIDTH 40,25
10 INPUT "ﾀﾝｶ";X
15 IF X=0 THEN GOTO 40
20 INPUT "ｽｳﾘｮｳ";Y
30 PRINT X*Y
32 Y$=INKEY$
33 IF Y$="" THEN GOTO 32
34 CLS
35 GOTO 10
40 END
```

ここでもう１つステートメントを追加しておきましょう。

行番号30をごらんください。ここでは２つの変数ＸとＹの中味をかけ算して、その結果をＰＲＩＮＴ文で表示していますが、演算は別のステートメントですませておくと、プログラム的には、よりすっきりした形になります。次の１行を追加します。

```
25 Z=X*Y
```

ＸとＹをかけ算して、その結果をＺという変数の中にしまっています（ＸとＹの積をＺに代入する、といいます）。

そうしますと行番号30の「Ｘ＊Ｙ」も「Ｚ」に変えておく必要がありますね。次のように変更しましょう。

```
30 PRINT Z
```

CTRL ＋ E

わざわざキーインする必要はありませんよ。カーソルをＸの上に重ね、Z キーを押します。するとカーソルは * の上に行きますから、CTRL キーを押しながら E キーを押してください。その行の、カーソル以下の部分が削除されます。

コントロールキー（CTRL）にはいろいろな使い方がありますが、この、ある行の不要な部分を削除する方法はよく使いますから、覚えておいてください。

3・9　変数の精度を明確に

私たちが「ＮＯＨＩＮ」というプログラムの中で使っている数値変数はＸ、Ｙ、Ｚの３つです。

35ページで変数の型について説明しましたが、コンピュータは、扱う変数の型宣言がなされていない場合は、すべて単精度型と見なして処理を進めます。処理速度は、整数型、単精度型、倍精度型の順で遅くなります。ほとんどの変数が整数型でしたら、ＤＥＦＩＮＴ文でＡからＺまでの変数を整数宣言しておいて、特定のものだけ単精度、倍精度の宣言をしておいたほうが、よほど効率がよいのです。

このプログラムでは、たとえば単価Ｘは100万円未満であるとして単精度を、数量Ｙは32767以下であるとして整数型を使うことにしますと、金額は100万円以上（つまり７桁以上の精度が必要）になる場合が考えられますので変数Ｚには倍精度型を使うことにします。

まず一度、すべての変数を整数宣言してしまいます。

```
2 DEFINT A-Z
```

このままですと変数ＸもＺも整数扱いになってしまいますから、あらためて、そのうしろに「！」「＃」をつけて単精度、倍精度の宣言をします。変更点と変更後のプログラムリストを次に示します。

●「Ｘ」と「Ｘ！」、「Ｙ」と「Ｙ＃」はまったく違う変数として扱われます。

```
10 INPUT "ﾀﾝｶ";X!
15 IF X!=0 THEN GOTO 40
25 Z#=X!*Y
30 PRINT Z#
```

```
1 CONSOLE 0,25,0,0:WIDTH 40,25
2 DEFINT A-Z
10 INPUT "ﾀﾝｶ";X!
15 IF X!=0 THEN GOTO 40
20 INPUT "ｽｳﾘｮｳ";Y
25 Z#=X!*Y
30 PRINT Z#
32 Y$=INKEY$
33 IF Y$="" THEN GOTO 32
34 CLS
35 GOTO 10
40 END
```

3・10　表を作ってみよう

この節では、プログラム「NOHIN」の実行結果を表にして画面表示するための下づくりをしますが、その前に、前ページのプログラムの行番号がふぞろいなので**RENUMコマンド**で整理しておきましょう。次のようにキーインすればよかったですね。

```
RENUM
```

プログラムリストは次のようになりました。

```
10 CONSOLE 0,25,0,0:WIDTH 40,25
20 DEFINT A-Z
30 INPUT "ﾀﾝｶ";X!
40 IF X!=0 THEN GOTO 120
50 INPUT "ｽｳﾘｮｳ";Y
60 Z#=X!*Y
70 PRINT Z#
80 Y$=INKEY$
90 IF Y$="" THEN GOTO 80
100 CLS
110 GOTO 30
120 END
```

RENUMコマンドを実行すると、GOTO文の飛び先行番号も自動的に変えてくれます。たとえば、前ページの行番号15のGOTO文の飛び先は40行でしたが、上の画面では120行になっていますね。

テキスト画面の座標

テキスト画面

私たちが現在見ているディスプレイ画面はテキスト画面といいます。これに対してグラフィック画面がありますが、ここではふれません。

キャラクタ

テキスト画面とは、「A」「B」とか「ア」「＃」「％」といったキャラクタが表示される画面のことをいいます。そして、テキスト画面に表示されるキャラクタの数は、2・6節で学んだWIDTH文によって決められます。

1画面のキャラクタ数＝(横の桁数)×(縦の行数)

たとえば、私たちのプログラムでは、WIDTH文によって横40桁、

縦25行にしてありますから、

キャラクタ数＝40×25＝1000(字)

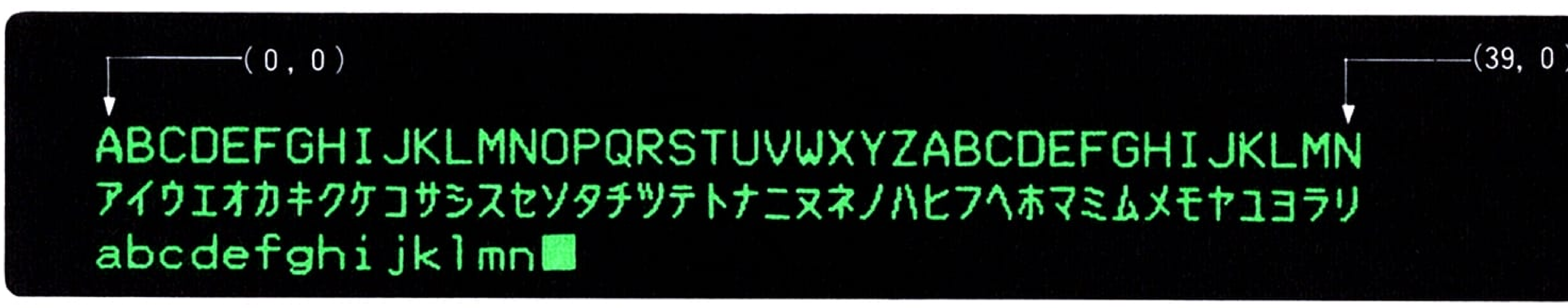
ということになります。

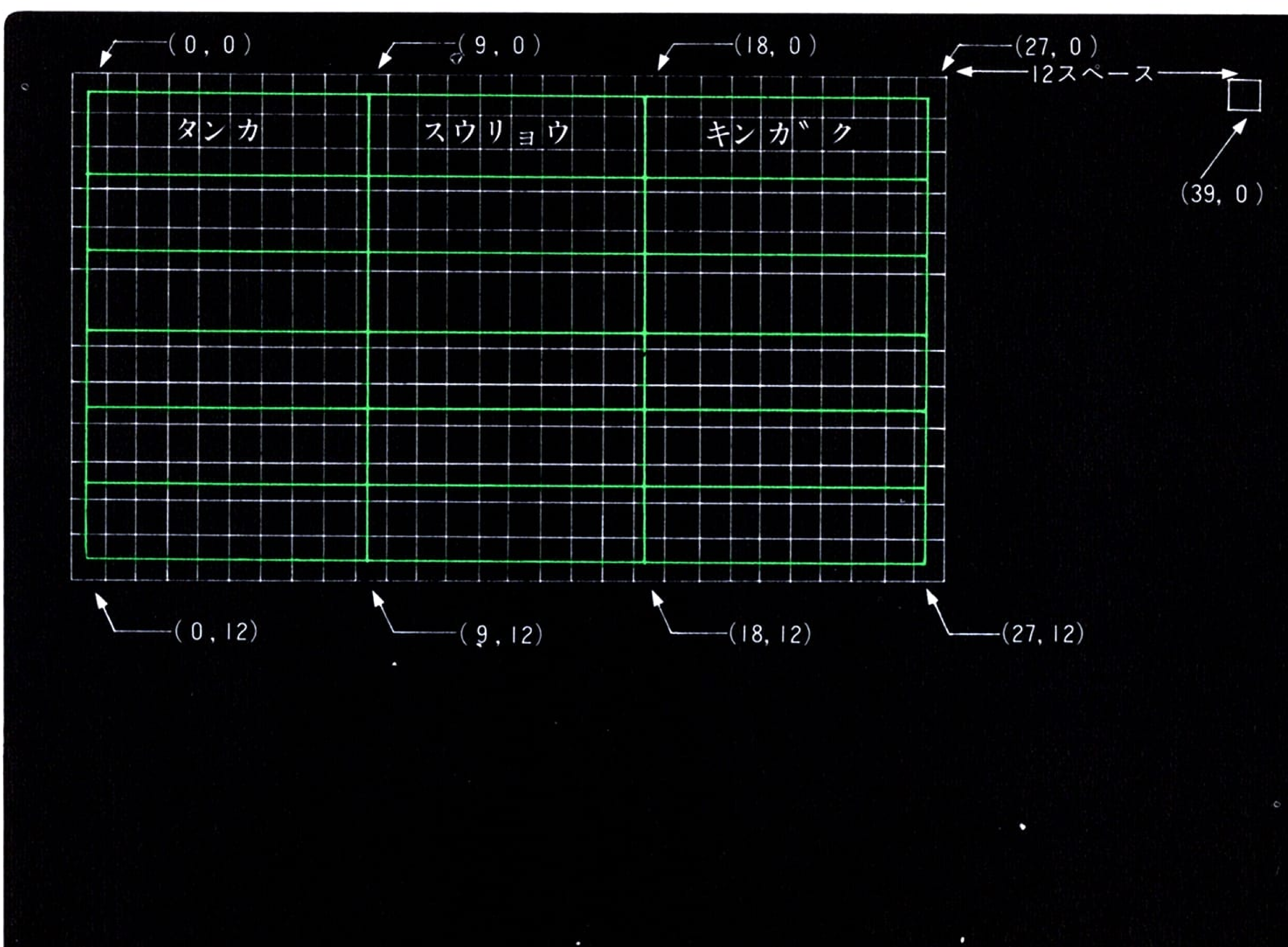
そして、この1000字分（40桁×25行の場合）の１画面上での位置が決められていて、これを**キャラクタ座標**といいます。キャラクタ座標は、画面の一番左側の列から何番目かをＸ座標、一番上の行から何行目かをＹ座標として次のようにあらわします。

キャラクタ座標

キャラクタ座標＝(Ｘ座標，Ｙ座標)

基準になる点（原点）は、ホームクリアキーで画面をクリアしたときにカーソルが点滅する位置で、座標は（1，1）ではなく(0，0)と表記します。一番上の行の２文字目でしたら(1，0)、一番右下すみは(39，24)となります。上３行を例にとると次の通りです。





表を画面に描く

前ページ下の図をごらんください。私たちがこれから作ろうとする表のレイアウトです。

ホームクリアキーで一度画面をクリアしてください。カーソルが点滅しているところが（0,0）の位置です。

次にグラフィックキャラクタで画面に前ページのような表（緑色の部分）を描いてください。たとえば、(0,0)の位置のキャラクタは、GRPH キーとテンキー側の7を押せばよいですね。(0, 2)の位置は GRPH キーとテンキー側の*キーです。

なれないうちは、グラフィックキャラクタをさがすのが大変ですが、がんばってみてください。この操作は、カーソル移動キーで描きたい場所にカーソルを動かし、どこから描いてもかまいません。ただ、ホームクリアキーだけは決して押さないようにしてください。せっかく描いた図が消えてしまいます。

見出しの文字は2行目に入れます。カナ キーをロックして書いてください。グラフィックキャラクタを表示するときは、カナ キーのロックをといてから入力しましょう。

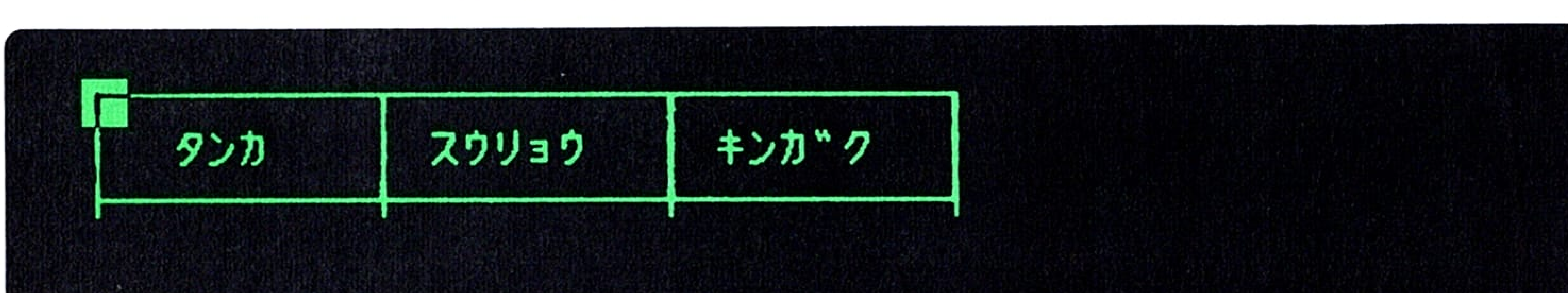
画面上の図をプログラムにする

さあ、完成しましたか？　もう一度確かめてください。

間違いありませんでしたら、この図をプログラムに変えてしまいましょう。画面表示は上の3行だけを使いますのでご諒承ください。

操作手順は次の通りです。

❶ まず、カーソルをホームポジション、つまり(0, 0)の位置に移動します。画面は次のようになっています。

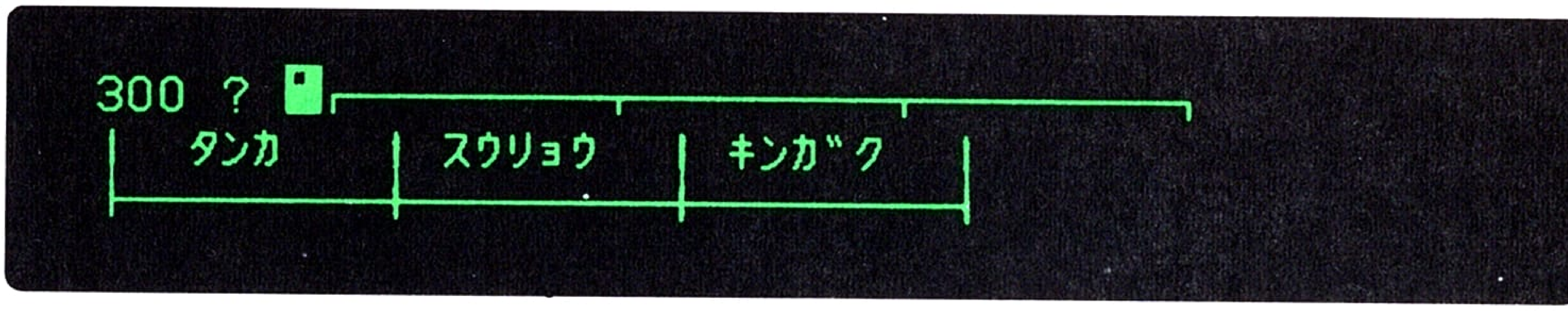


❷ 次に、この位置で SHIFT キーを押しながら INS キーを一度押します。これでインサートモードになりました。

❸ 次のようにキーインしてください。

3 0 0 （スペース） ? （スペース） ″

画面は次のようになりましたか？

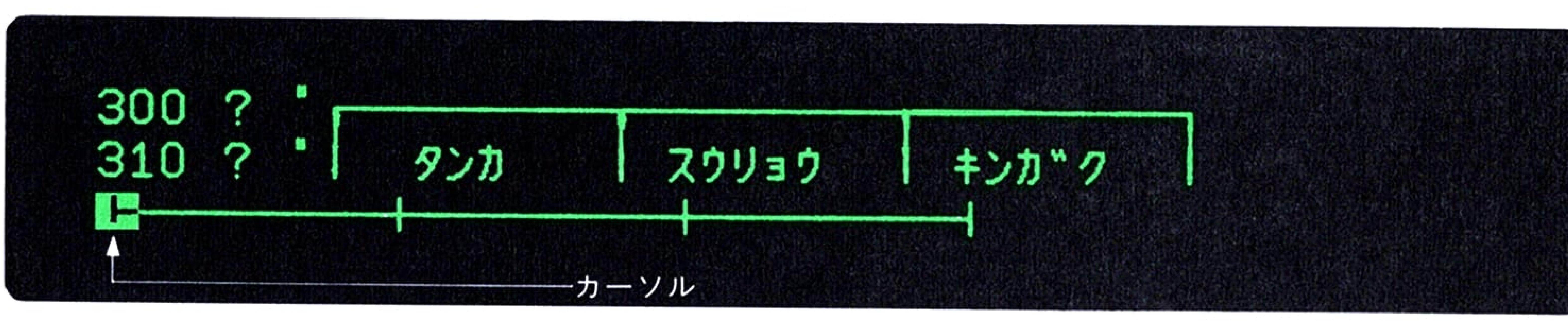


インサートモードで6字分挿入されましたので、表の一番上の線がその分だけ右へ寄りました。

❹ 上の画面の状態で ⏎ キーを押してください。カーソルは2行目の左はじに行きます。

❺ 画面の「300」という数字は行番号になります。

❻ 2行目は行番号310にすることにして、❷からの操作をくりかえします。



いかがですか？　もう要領はのみこめたと思います。

それでは、表の一番下の線のある13行目まで、同じようにやってみてください。

クォーテーションマークまでインサートしたら、必ず ⏎ キーを押してくださいよ。といっても、あまりかたくならないで気軽な気持でお願いします。間違えても、もう一度やりなおせばよいのですから。また、そのほうがかえってスクリーンエディトの力を高めるかもしれません。

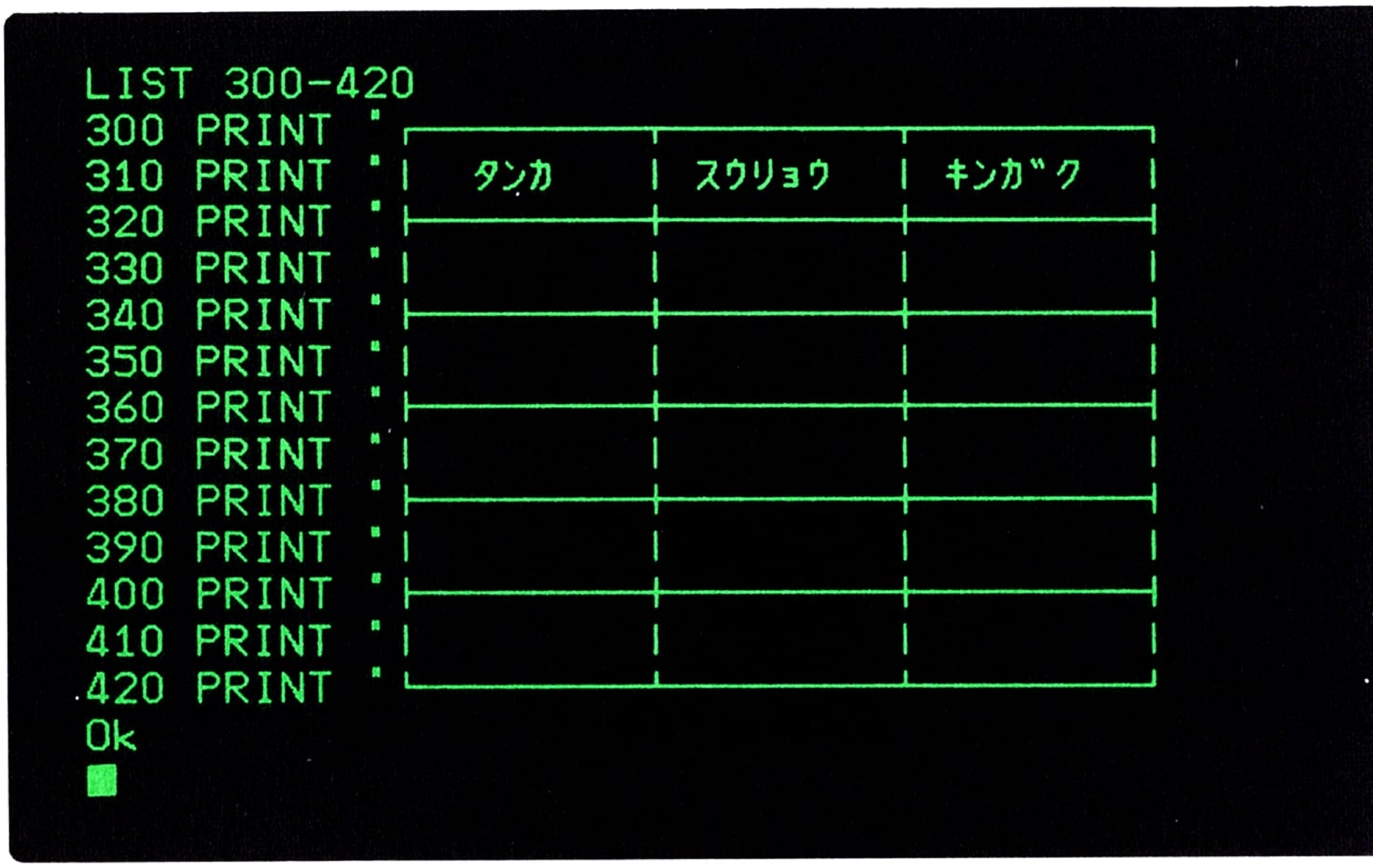
信じられないかもしれませんが、これで、私たちが画面に描いた表がプログラムになってしまったのです。ホントかウソか、とにかくためしてみましょう。

ホームクリアキーで画面をクリアしてから、次のようにキーインします。

```
LIST 300-420 ⏎
```

これは「行番号300から行番号420までのプログラムリストを表示しなさい」という命令です。60ページでは1つの行番号だけ指定しましたが、このように**マイナス記号によって2つの行番号を結ぶと、その間のすべての行を表示してくれます。**行番号をまったく指定しないと、プログラムの先頭から末尾までのステートメントがすべて表示されます。

リターンキーを押すと、画面は次のようになりました。



入力は「?」でしましたが、「PRINT」と変わっていますね。

ここで1つ補足説明をしておきます。

PRINT文を用いて文字列（ストリング）を画面に表示する場合、私たちは34ページで、表示したいストリングをダブルクォーテーションでくくっていました。ところが、上のリストでは、どの行も右側のダブルクォーテーションが入っていません。

このように**PRINT文では、うしろのダブルクォーテーションは省略することができます。**ただし、そのうしろにコロンでステートメントをつなげる（マルチステートメント）場合には省略することができません。

3・11　画面のレイアウト

まず、前節で作った作表プログラムを実行してみましょう。

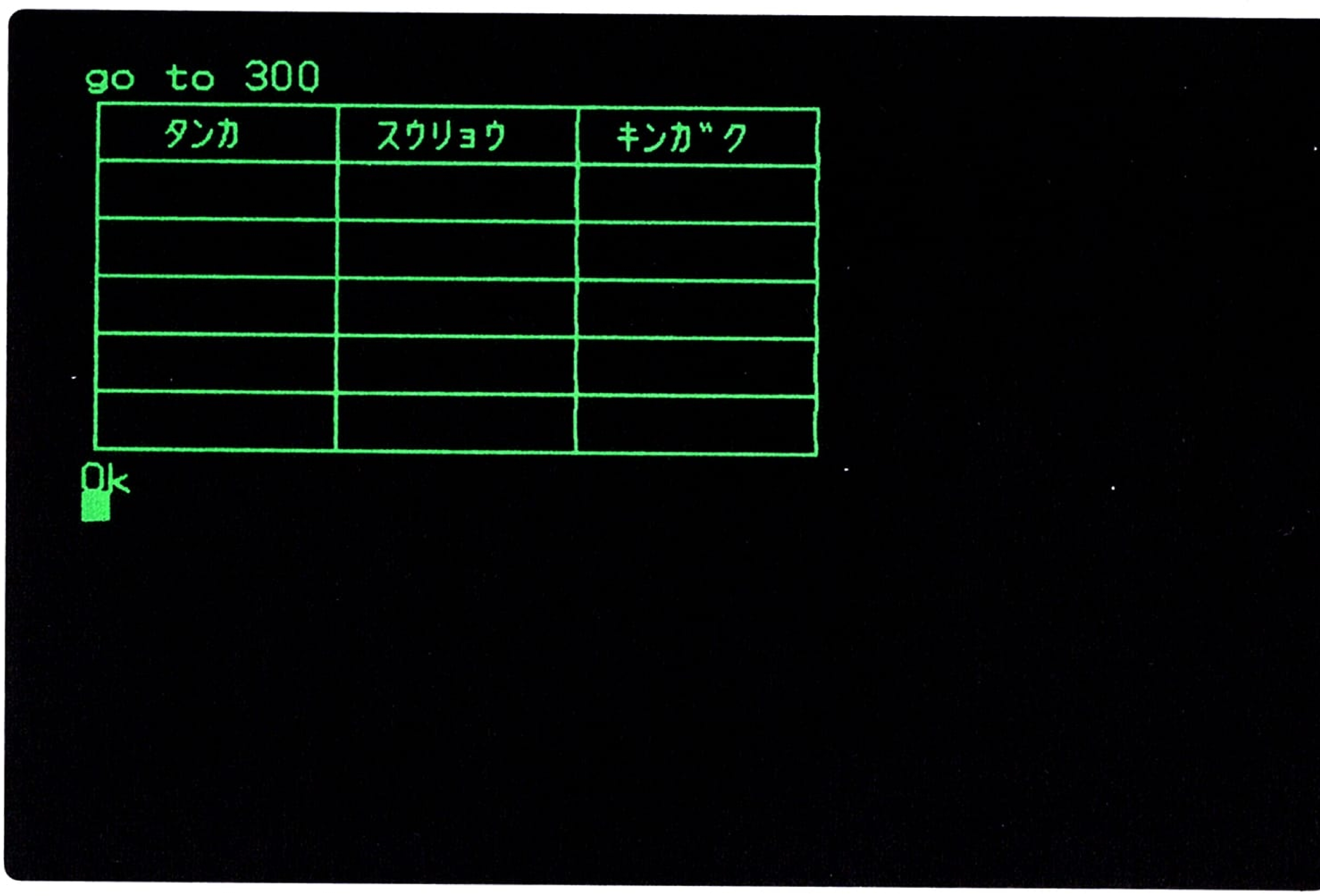
いままでプログラムを実行するときは (f・5) キーを押していましたが、ここでは画面をクリアして次のようにキーインしてください。

```
GOTO 300 ↵
```

「GOTO」は (f・3) キーに定義されていますから、これを使えば楽ですね。

上のようにGOTO文により行番号を指定すると、その行のステートメントから処理を実行してくれます。

画面はどうなりましたか？　次のようになったでしょうか？



確かに表はうまく表示されましたが、画面の左すみのほうに寄ってしまって少々アンバランスです。これをもう少し中央寄りに移動したいものですね。

任意の位置に表示する

ここで登場するのが**LOCATE（ロケート）文**です。　LOCATE

これは、67ページで説明したキャラクタ座標を与えると、画面上の

その位置から、キャラクタとか数値などを表示してくれます。
書式は次の通りです。

```
LOCATE X,Y
```

X：表示位置のX座標
Y：表示位置のY座標

座標についてはもうおわかりですね。横40桁、縦25行の画面モードでは、Xの値は0から39まで、Yの値は0から24まで指定できます。**このＬＯＣＡＴＥ文の次にＰＲＩＮＴ文をおけば、その指定位置にキャラクタを表示できます。**

画面をクリアして、次のステートメントをダイレクトモードで入力してみてください。

```
LOCATE 19,2:PRINT "AAA"
                   AAA
Ok
```

(19, 2)
左から20桁目、上から3行目の位置から表示

スリーＡが所定の位置に表示されました。

さて私たちは、前ページの表をもう少し中央寄りに移動したいわけですが、行番号300から420までのＰＲＩＮＴ文をどこから表示させたらよいでしょうか？

表の左上すみのX座標は0になっています。

ここで67ページの画面をごらんください。表の右上すみと、画面の右上すみとの間は12スペースあいています。ですから、X座標を12の半分、つまり6字分だけ全体を右へ移動すれば、ちょうど中央になります。

したがって、表の一番左の列の表示位置のX座標は6になります。

ではY座標はどうしましょうか？　67ページの表をちょうど上下中央にもってくると、視覚的な錯覚があり、やや下のほうに表示されているように見えてしまいます。ですから、安定感をもたせるために、少し上のほうに表示するようにします。ここでは全体を4行分下げて5行目から表示することにします。するとY座標は4になりますね。5ではないですよ。一番上の行が0行ですから……。

表示位置については人間それぞれ好みがあります。みなさんもあとで、自分の好きなように変えてみてください。

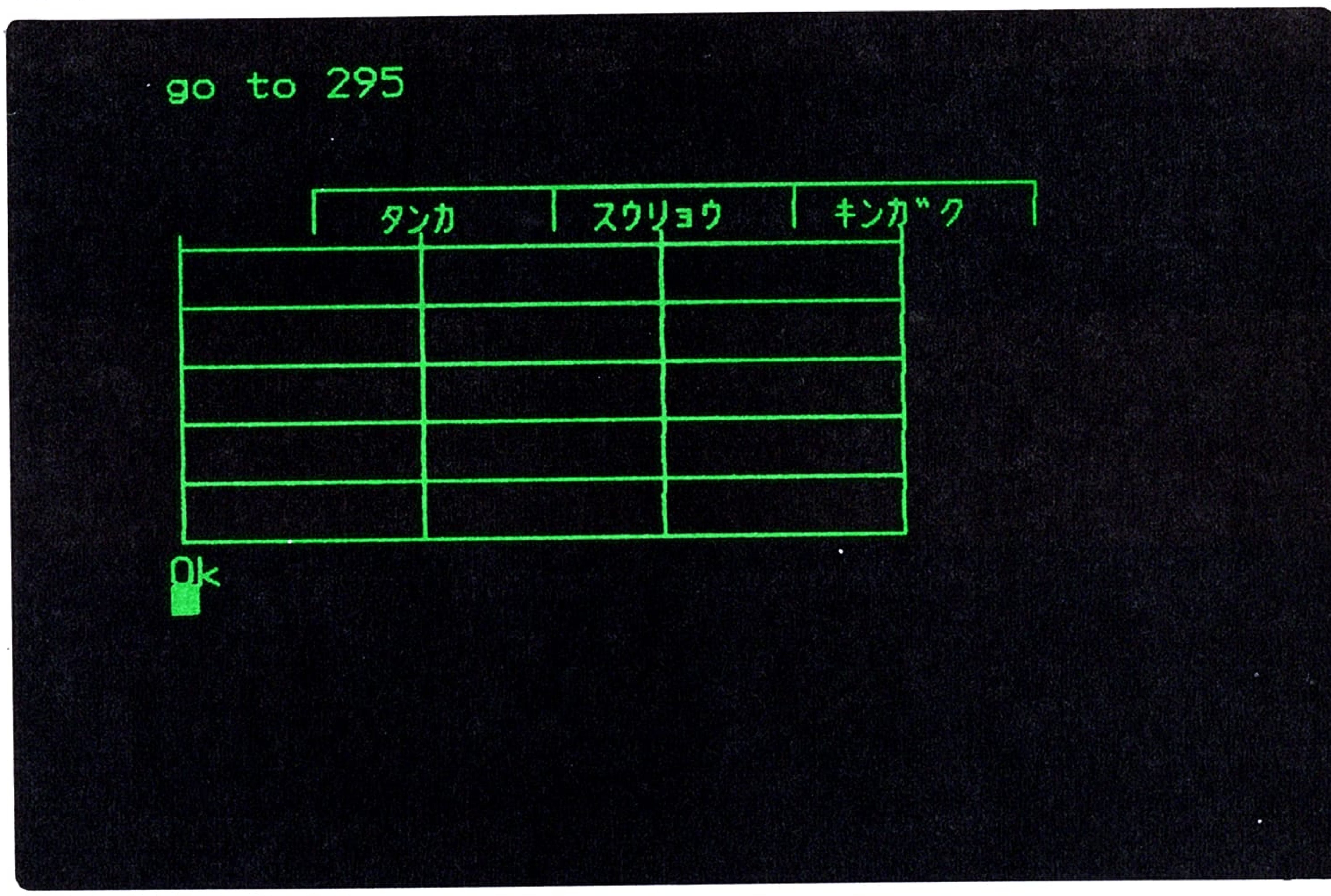
これでX軸方向、Y軸方向の全体の移動量が決まりました。ところが、全体を移動する場合、X座標はどの行も左はしがそろっているわけですから6でよいのですが、Y座標は行が下へさがるにつれて1ずつふえていきます。表の1行目、2行目について考えてみましょう。

ＬＯＣＡＴＥ文をＰＲＩＮＴ文の前に入れなければなりませんから、行番号をそれぞれ295、305として次のようにキーインします。

```
295 LOCATE 6,4
305 LOCATE 6,5
```

Y座標は上の行より1だけふやす

キーインできましたか？　できましたら、ここで実行してみましょう。今度はＧＯＴＯ文で行番号295に飛ばせばよいわけですね。



上の2行分がちょうどいい位置に表示されましたね。

同じようにして3行目以降もＬＯＣＡＴＥ文を使って移動すればよいのです。そこで、行番号315、325、335、345、355、365、375、385、395、405、415をキーインすることになりますが、これはわざわざバカていねいにキーインするまでもなく、ずいぶん楽な方法があります。

いいですか？　**どの行も「ＬＯＣＡＴＥ　6,」の部分は同じで、異なるのは行番号とY座標だけです。ですから、この、違う部分だけ変えればよいのです。**

出発点にするステートメントとして305行を使うことにします。ホ

ームクリアキーで画面をクリアして、ＬＩＳＴコマンドで305行を画面に出してください。ＬＩＳＴは (f・4) キーを使いましょうか。

```
list 305
305 LOCATE 6,5
Ok
█
```

そうしましたら、カーソルを「305」の「0」に重ね、 [1] のキーを押します。カーソルは「5」のところでブリンクしています。

```
315 LOCATE 6,5
```

次にカーソルを一番うしろの「5」に重ね [6] のキーを押し、リターンキーを押してください。画面は次のようになっています。

```
list 305
315 LOCATE 6,6
█
```

「5」を「6」に変えました

これで、305行を使って315行を作ることができました。305行は消えてしまったように思えますが、コンピュータにはちゃんと記憶されています。行番号305から315までのリストを画面に表示してみましょう。

```
list 305-315
305 LOCATE 6,5
310 PRINT "|   ﾀﾝｶ     | ｽｳﾘｮｳ    | ｷﾝｶﾞｸ    |
315 LOCATE 6,6
█
```

やはり消えていませんでした。

今度は、いま画面に表示されている315行を使って325行を作ります。わざわざＬＩＳＴコマンドで315行を画面にだす必要はありませんよ。いま説明した手順でやってみてください。ただし、修正が終ったら必ずリターンキーを押してください。そうしないと修正したことになりません。

次は325行をもとにして335行を、というように私たちが必要としている415行までのＬＯＣＡＴＥ文を作ってください。

さあ、これで、作表プログラムは完成です。リストをとって確認してみましょう。

```
list 295-420
295 LOCATE 6,4
300 PRINT "┌─────────┬─────────┬─────────┐
305 LOCATE 6,5
310 PRINT "│  ﾀﾝｶ    │ ｽｳﾘｮｳ   │ ｷﾝｶﾞｸ   │
315 LOCATE 6,6
320 PRINT "├─────────┼─────────┼─────────┤
325 LOCATE 6,7
330 PRINT "│         │         │         │
335 LOCATE 6,8
340 PRINT "├─────────┼─────────┼─────────┤
345 LOCATE 6,9
350 PRINT "│         │         │         │
355 LOCATE 6,10
360 PRINT "├─────────┼─────────┼─────────┤
365 LOCATE 6,11
370 PRINT "│         │         │         │
375 LOCATE 6,12
380 PRINT "├─────────┼─────────┼─────────┤
385 LOCATE 6,13
390 PRINT "│         │         │         │
395 LOCATE 6,14
400 PRINT "├─────────┼─────────┼─────────┤
405 LOCATE 6,15
410 PRINT "│         │         │         │
415 LOCATE 6,16
420 PRINT "└─────────┴─────────┴─────────┘
Ok
█
```

上の画面とよく見くらべて、間違いなかったら実行してみてください。画面をクリアしてから、GOTO文で295行へ飛ばせばいいですね。71ページの画面の表が中央に移動していれば正解です。

もし、ちぐはぐになっていたりしたら、座標があっているかどうか確認してください。

上のリストは1画面にはおさまりませんから、どんどん上へスクロールしてしまいます。どこか途中で1時的に止めたいときは、コントロールキー CTRL を左小指で押さえ、左人差指で S キーを押さえます。もう一度 S キーを押せばLISTが続行されます。

CTRL ＋ S

3・12　表に数値を書き込む

何か完成したら、それでもものたりなくて次は何をしようか、と考えるのが人間です。私たちも次のステップへ進みましょう。いま作った表の中に、単価とか数量とか金額を書き込むのです。

サブルーチン化する

もう一度、単価、数量を入力する部分のプログラムをふりかえってみましょう。66ページをごらんください。

30行で単価を入力し、50行で数量を入力し、60行で金額を計算し、それを70行で画面に表示しています。そして現在の私たちは、これらの数値を先ほど作った表の中に表示させたいのです。ということは、1回目の単価、数量の入力が終り、60行の金額計算が終ったら、画面をクリアして表を画面に表示し、各数値をそこにはめ込むようにすればよいわけです。ですから、この段階で、295行へ処理を飛ばせばよいのです。

処理をほかの行に移す場合、私たちはいままでＧＯＴＯ文を使っていました。しかしこのＧＯＴＯ文は、指定されたところへ飛ぶのですが、飛び先から戻ってきません。

私たちは、単価、数量の入力をまだまだ続けたいわけですから、作表部分へ飛んでいって、そこでこのプログラムの処理が終ってしまったら困るわけです。ここは、どうしても帰ってきてもらわなければなりません。

ＧＯＳＵＢ

そこで登場するのが**ＧＯＳＵＢ(ゴーサブ)文**です。

ＧＯＴＯ文には「行け！」というひびきが、一方「ゴーサブ」文には「サブちゃん行って来いよ」という感じがあります。つまり、この忠実なサブちゃんはもとの場所へ戻ってくるのです。

ＧＯＳＵＢ文の書式は次の通りです。

GOSUB	(行番号 またはラベル)	RETURN

このように、**ＧＯＳＵＢ文は必ずＲＥＴＵＲＮ文と対で使われます。**

行番号を指定すると、その行番号へ飛び、ＲＥＴＵＲＮ文に出会ったら、飛び出したＧＯＳＵＢ文のすぐうしろに戻ってきます。つまり、引き続きＧＯＳＵＢ文のうしろのステートメントから処理が実行されることになります。

ラベル

ラベルを使う場合も考え方はまったく同じで、行番号がこれにとってかわっただけです。

ここで、GOTO文とGOSUB文の違いを確認してみましょう。

いまみなさんのコンピュータに記憶されているプログラムを少しこわしますから、カセットテープかディスケットにセーブしておいてください。

セーブの方法はわかりますね。ファイル名は「NOHIN」です。わからない人は3・3節をごらんください。

カセットテープをご使用の方は必ず、セーブできたかどうかの確認(ベリファイ)をしてみてください。

セーブできましたら、次のようにキーインしてください。大文字でも小文字でもかまいません。

```
DELETE 40-420 ↵
```

そうしたらLISTコマンドでリストをとってみてください。

```
DELETE 40-420
Ok
list
10 CONSOLE 0,25,0,0:WIDTH 40,25
20 DEFINT A-Z
30 INPUT "ﾀﾝｶ";X!
Ok
█
```

40行以降のステートメントがすべて消えてしまいました。

DELETE

このように**DELETE(デリート)**は、削除したい範囲の行番号と行番号をマイナス符号で結べば、たちどころに消してくれます。

さて、この状態で次のようにキーインしてください。

```
40 GOTO 1000 ↵
1000 PRINT X! ↵
```

この2つの行を追加することによって、プログラムの意味は「30行で単価を入力し、1000行へ飛んで単価を画面に表示する」ということになりました。

それでは (f・5) キーを押して、このプログラムを実行してみてください。1回入力しただけでOkがでてしまいますね。

では次のステートメントをキーインしてください。

```
40 GOSUB 1000 ⏎
50 GOTO 30 ⏎
1000 PRINT X! ⏎
1010 RETURN ⏎
```

行番号40、1000はすでに入力ずみの行番号と重複しますが、あとからキーインしたものが優先されて残ります。

画面クリアしてリストをとってみましょう。

```
list
10 CONSOLE 0,25,0,0:WIDTH 40,25
20 DEFINT A-Z
30 INPUT "ﾀﾝｶ";X!
40 GOSUB 1000
50 GOTO 30
1000 PRINT X!
1010 RETURN
Ok
█
```

次に (f・5) キーを押して実行してみます。

```
run
ﾀﾝｶ? 200
 200
ﾀﾝｶ? █
```

どうですか？　何回でも入力できますね。

これはＧＯＳＵＢ～ＲＥＴＵＲＮが効いているからです。行番号40のＧＯＳＵＢ文によって1000行へ飛び、単価を画面に表示して1010行のＲＥＴＵＲＮ文によってＧＯＳＵＢ文のすぐうしろに帰ってきて、次のステートメント、つまり50行を実行するわけです。そして30行のＧＯＴＯ文により、ふたたび行番号30から処理をくりかえすわけです。

これでＧＯＳＵＢ文については理解していただけたと思います。

それでは (STOP) キーを押してプログラムにブレークをかけ、私たちがいま作成中のプログラム「ＮＯＨＩＮ」をロードしてください。ロード方法のわからない方は3・3節をごらんください。

ロードできましたか？　ＬＩＳＴコマンドで確認しておきましょう。

それでは本題に戻ります。

先ほどＧＯＳＵＢ文のイメージとして、「サブちゃん行って来い」と述べましたが、「ゴーサブ」の「サブ」は実際には**サブルーチン**の略です。これは「メインルーチン」に対する用語で、メインルーチンに付随した仕事（処理）を何度もくりかえすような場合、これをサブルーチン化しておきますと、プログラムも短かくなり能率的です。

サブルーチン

私たちは、「作表」と「表中への数値表示」の部分をサブルーチンにすることにしましょう。

もう一度メイン部分のプログラムを次に示します。

```
list 10-120
10 CONSOLE 0,25,0,0:WIDTH 40,25
20 DEFINT A-Z
30 INPUT "ﾀﾝｶ";X!
40 IF X!=0 THEN GOTO 120
50 INPUT "ｽｳﾘｮｳ";Y
60 Z#=X!*Y
70 PRINT Z#
80 Y$=INKEY$
90 IF Y$="" THEN GOTO 80
100 CLS
110 GOTO 30
120 END
Ok
```

上の画面を見ますと、行番号60の金額計算で１回の処理が終っています。ですから、ここから作表サブルーチンに飛ばし、次に各数値を書き込めばよいわけです。

行番号65として、ＧＯＳＵＢ文をキーインします。

```
65 GOSUB *SAKUHYO
```

ここでは、飛び先を行番号でなくラベルで指定してみました。ラベルは「＊(アスタリスク)」と具体的な処理名で構成されます。長いプログラムになりますと、行番号指定では何をするルーチンかすぐに判断できませんが、このようにラベルを使っておけば一目瞭然です。

１行入力したら必ずリターンキーです。以後、 のマークは省略しますが、忘れないように注意してください。

次に作表部分をサブルーチン化するわけですから、この部分にＧＯＳＵＢの飛び先がわかるようにラベルをつけます。行番号は290としておきます。次のようにキーインします。

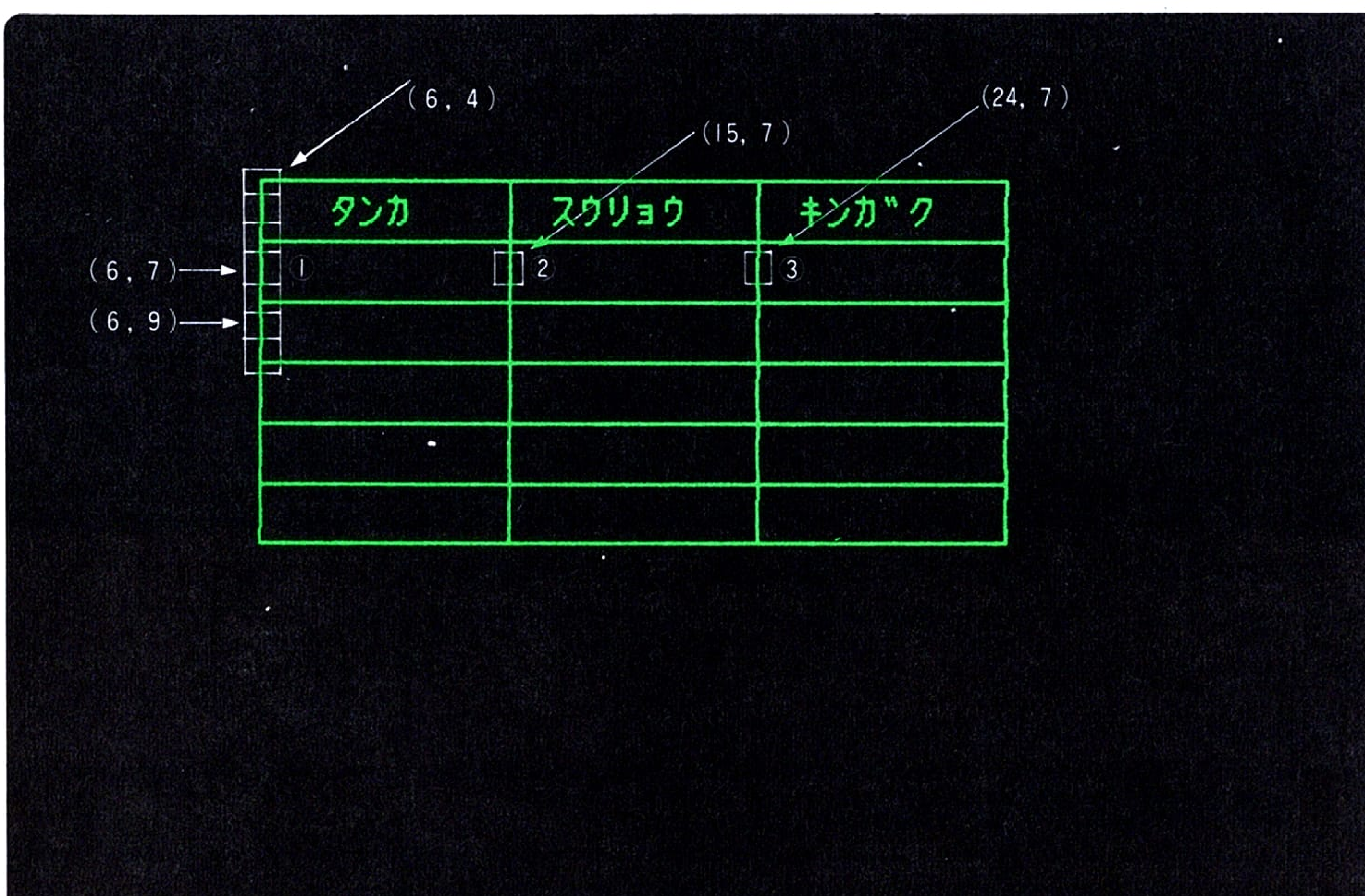
```
290 *SAKUHYO ↵
```

ラベル名のスペル（つづり）を間違えたりすると実行したときにエラーになったりしますから注意してください。これは、よくおかしがちなミスです。

さて、65行のＧＯＳＵＢ文によってこのルーチンに飛んできますと、75ページの画面にあるように、行番号295から420までの作表処理を実行します。ですから、この処理が終った段階で、つまりこのうしろに数値表示のプログラムを追加すればよいわけです。

これにはＰＲＩＮＴ文を使いますが、表示位置はＬＯＣＡＴＥ文で指定しなければなりません。

次の図を見てください。



私たちのプログラムでは、表は上の図のような位置に表示されています。ですから、単価、数量、金額をそれぞれ❶、❷、❸の位置から表示させるようにすればよいわけです。❶、❷、❸の座標は次のようになっています。

❶→(7，7)　❷→(16，7)　❸→(25，7)

わかりやすくするため単価についてだけ考えてみましょう。

単価は行番号30で入力しましたから、変数X！の中にしまわれています。もうわかりますね。ＬＯＣＡＴＥ文で位置を指定して、ＰＲＩＮＴ文によってこの値を画面に表示すればよいのです。次のようにキーインします。

```
430 LOCATE 7,7:? X!
```

ここでは、コロンでつないでマルチステートメントにしてみました。マルチステートメントについては62ページで説明しましたね。

同じように、数量、金額についてもキーインしましょう。

```
440 LOCATE 16,7:? Y
450 LOCATE 25,7:? Z#
```

さて、金額用の変数の内容はこのサブルーチンで表示することになりましたので、行番号70のＰＲＩＮＴ文は不要になります。削除してしまいましょう。77ページで削除については説明しましたが、１行だけの場合にはもっと簡単な方法があります。削除したい行番号をキーインし、リターンキーを押すだけでよいのです。

[7] [0] [↵] ← 行番号70を削除（デリート）

これで Ok です。うたがわしいと思う方は、ＬＩＳＴコマンドで行番号70を表示させてください。何もでてきませんから……。

次に一番大切なこと、サブルーチンでの仕事が終ったら、来たところへ戻るためにＲＥＴＵＲＮ文をつけなければなりません。仕事で海外に行ったのに、何が楽しかったのか帰ってこない、これでは困りますね。ちゃんと帰って、メインとなるこの日本で仕事を続けなければ。次の行を追加します。

```
460 RETURN
```

サブルーチンができあがりました。実行してみてください。

何か気づいたことはありませんか？　そうです。表示される数値はいつも一番上の欄だけで下の４つは空白です。これは次節で解決しましょう。それからもう１つ。入力した値が画面左上に残ったままになっています。次の行を追加してクリアすることにしましょう。

●金額が100万以上になりますと表の右端が切れてしまいますが、これは数値のうしろに空白が１字分とられるからです。この処理はあとで説明します。

```
292 CLS
```

3・13　配列を使おう

どうもコンピュータの勉強はかたくるしいことばかりで困りますねえ。まあ、これも仕事の能率化のためとあきらめて、がんばりましょう。

配列とは？

私たちはこれまで、いくつか変数を使ってきました。X、Y、Zなどがそうですね。そしてこの変数は、英数字で表現するかぎり、制約は多少ありますが、ずいぶん自由に使えます。

たとえば、ある小学校のクラスの生徒たちの成績用の変数として次のようなものを使ったとします。

A1, A2, A3, A4,……A15, A16,……A49, A50

Aに続く数字（1,2,3,……50）は出席簿順としますと、A1には出席簿で1番（成績ではありませんよ）目の新井さんの成績が、A16には佐藤さんの成績が、そしてA50には出席簿では一番ドンジリの渡辺さんの成績が入る、ということになります。

これを配列の表現に変えますと、次のようになります。

A(1), A(2),……A(15),……A(49), A(50)

A1もA(1)も、またA15もA(15)も同じようにその中に成績を格納することができます。配列も一種の変数なのです。

それなら、「配列はカッコが多いだけめんどうじゃないか」ということになりかねませんが、これがぜんぜん違うのです。その便利さは徐々にわかると思いますので、ここでは考え方だけ説明しておきます。

最初のA1、A2、A3……は、Aというアルファベットが共通していても、それぞれはまったく別ものなのです。いわば国民に背番号をつけたようなもので、個々のつながりはあまり密接ではありません。

ところが一方、配列のほうは、Aという文字が1つのファミリーを表しているのです。そして、人それぞれの容姿、性格が違うように、格納されている内容こそ違え、同じファミリーという分離しがたいものがあります。このファミリー名に相当するのが**配列名**で、いまの場合は、配列Aという呼び方をします。カッコ内の数字は**添字**といいますが、まあ、配列Aの中の順番だくらいに考えてください。

配列名
添字

それではここで、配列も普通の変数と同じに扱えることをためしてみましょう。2・3節と同じことを行ってみます。くわしい説明は必要ないと思いますから、次の画面を見ながらダイレクトモードで各自やってみてください。

```
A(1)=5:A(2)=3
Ok
? A(1)
 5
Ok
? A(2)
 3
? A(1)+A(2)
 8
Ok
█
```

2・3節のXをA(1)に、YをA(2)にたとえてみましたが、X、Yを使ったときとまったく同じ結果が得られます。

これで配列A(1)もA(2)もそれぞれ1つの変数と考えてよいことがご理解いただけたと思います。

配列の宣言

やはりダイレクトモードで次のようにキーインしてみてください。

```
A(10)=2 ↵
A(11)=4 ↵
```

いかがですか？　最初はだいじょうぶですが、A(11)の場合にエラーメッセージが表示されましたね。

```
A(10)=2
Ok
A(11)=4
Subscript out of range
Ok
█
```

〈注〉エラーメッセージはディスク内蔵タイプとモデル10で異なりますが、本書では前者のメッセージを採用します。モデル10の方はマニュアルをごらんください。なお、表現が違うだけでエラーの原因は同じです。

これはなぜかといいますと、配列は初期状態では添字10まで、添字は1ではなく0から始まりますから、A(0)からA(10)まで、計11個の場所きり確保されていません。ですからA(11)は12番目の要素になりますから、Subscript out of range(許される配列の範囲を超えていますよ)というメッセージが表示されたわけです。

そのため、もっとたくさんの要素を使いたいときは、プログラムのはじめのほうで宣言をしておきます。たとえば100だけ確保したければ、次の書式によって宣言します。

DIM

```
DIM A(100)
```

これはDIM(ディム)文といわれ、DIMENSIONの略です。こうしておけば、A(0)からA(100)までの101個の配列変数を使うことができます。

配列変数のメリット

普通のA1, A2,……A10などの変数に適当な数値を代入してみましょう。

```
A1=5:A2=5:A3=5:A4=5:A5=5:A6=1:A7=1:A8=1:A9=1:A10=1
Ok
■
```

これでA1からA5までに5が、A6からA10までに1が入りました。それでは、これらの変数の中をクリアする(ゼロにする)にはどうしますか? そうです。もう一度次のように、こまめにキーインしていかなければなりませんね。

```
A1=0:A2=0:A3=0:A4=0:A5=0:A6=0:A7=0:A8=0:A9=0:A10=0 ↵
```

では配列の場合はどうでしょうか。まず、各要素に数値を代入します。ここまでは上のケースと同じです。

```
A(1)=5:A(2)=5:A(3)=5:A(4)=5:A(5)=5:A(6)=1:A(7)=1:A(8)=1:A(9)=1:A(10)=1 ↵
```

さて、ここからの手間がだいぶちがいます。

FOR〜NEXT

結論から申し上げますと、**FOR〜NEXT(フォア〜ネクスト)文**を使います。次ページのように入力してください。

いま、あなたのコンピュータにはプログラム「NOHIN」が入っていますが、かまいません。あとで、このテテト部分は削除しますから……。

```
1 FOR I=1 TO 10
2  A(I)=0
3 NEXT I
4 END
```

行番号1のＦＯＲ文によって変数Ｉを１から10まで変化させていますが、最初はＩ＝１の状態で行番号２に処理が移ります。このときＡ(Ｉ)はＩが１ですからＡ(１)と表せます。ですから行番号２のステートメントは

Ａ(１)＝０

と考えられます。つまり、配列Ａ(１)の中味がクリアされたことになります。次に処理は行番号３のＮＥＸＴ文に移り、「ＮＥＸＴ　Ｉ」つまり「次のＩへ」ということで行番号１へ戻ります〈注〉。

〈注〉上のプログラムの実際の流れは
1 → 2 → 3
と進み、次に行番号１へ戻らず行番号２へ行きます。そして、(２→３)の処理を９回くり返してループを抜けます。しかし、理解のためには行番１と３の間を10回ループすると考えたほうがよいでしょう。

すでにＩ＝１の場合は処理がすんだわけですから、今度はＩは２になります。同じように行番号２で

Ａ(２)＝０

となり、２番目の要素もクリアされました。

そして、また「次のＩへ」というわけで、Ｉ＝３の場合がくりかえされます。

このようにして、Ｉ＝10となるまで、行番号１と行番号３の間の処理がくりかえされます。このくりかえしを**ループ**といい、Ｉのことを**ループ変数**といいます。

ループ
ループ変数

Ｉ＝10のときの処理がすみますと、このＦＯＲ～ＮＥＸＴループをぬけだし、行番号４に処理が移り、ＥＮＤ、つまりプログラムの実行が終ります。

それでは、Ａ(１)からＡ(10)までの各要素がほんとうにクリアされたかどうかためしてみたいと思います。

次のステートメントを入力してください。

```
4 FOR I=1 TO 10
5 PRINT A(I);
6 NEXT I
7 END
```

（5行目の「;」：セミコロン）

前ページの一番下のステートメントは入力してありますか？　まだの人はダイレクトモードで入力してください。

さあ、そうしたら実行してみましょう。画面のクリアはしてもしなくてもかまいません。 (f・5) キーを押してください。



0が10個横に並びましたね。一番左がA(1)の、一番右がA(10)の内容です。つまり、各要素がすべてクリアされました。

ここでアドバイスを1つ。

行番号5でA(I)のうしろに「;(セミコロン)」をつけました。これをつけておくと、次の結果を前の結果に続けて表示してくれます。*

上の例でいきますと、符号分として1スペースとられ、そのうしろにすぐ、最初の0が表示されます。次に、数値を画面に表示した場合、そのうしろに自動的にスペースが1つ入ります。そして、2番目の0の符号分のスペースが入り、続いて0が表示されます。つまり、上の画面の0と0の間は2スペースあいています。

＊セミコロンのかわりにコンマを使いますと、2つ目の結果はX座標14のところから出力されます。ためしてみてください。

数値表示の場合、そのうしろにスペースが1つ入るといいましたが、81ページの実行結果を思い出してください。金額が100万以上になると、符号分で1字分、100万は7桁ですから、計8字分必要になります。そして、さらにそのうしろにスペースが入りますから、このスペースによって表の右端が消えてしまったのです。

行番号5で、セミコロンをつけないときは、結果がタテに10個表示されます。PRINTには改行機能がありますから、前にセミコロンがなければ、改行して表示することになります。

表に数値を埋める

それでは、メインテーマに戻りますが、その前に、行番号1から行番号7までは不要ですので削除しておきましょう。

```
DELETE 1-7 ⏎
```

私たちの作った表は、数値を埋める欄が5行分あります。ですから、この5行分のデータがたまるまで、これらのデータを、配列の中にしまいこんでおくことにします。

単価用の配列をA!、数量用の配列をB、金額用の配列をC#としておきましょう。扱う数値の精度に配列名もあわせることに注意して

ください。

これらの配列に、5行分のデータ、つまり単価、数量、金額についてそれぞれ5つずつのデータを次のようにしまっておきます。

1つ目のデータ ➡ A!(1), B(1), C#(1)
2つ目のデータ ➡ A!(2), B(2), C#(2)
3つ目のデータ ➡ A!(3), B(3), C#(3)
4つ目のデータ ➡ A!(4), B(4), C#(4)
5つ目のデータ ➡ A!(5), B(5), C#(5)

このようにして、5行分のデータがたまったら、作表サブルーチンへ飛ばして表示させるようにします。

ここでメイン部分のリストを画面に表示してください。

```
list 10-120
10 CONSOLE 0,25,0,0:WIDTH 40,25
20 DEFINT A-Z
30 INPUT "ﾀﾝｶ";X!
40 IF X!=0 THEN GOTO 120
50 INPUT "ｽｳﾘｮｳ";Y
60 Z#=X!*Y
65 GOSUB *SAKUHYO
80 Y$=INKEY$
90 IF Y$="" THEN GOTO 80
100 CLS
110 GOTO 30
120 END
Ok
█
```

まず、添字は5までしか使いませんから、配列の宣言は必要ないのですが、添字が10以下でも宣言をしておきますと、必要最小限のメモリを確保することになりますから、多少なりとも効率がよいことになります。また、配列宣言の習慣をつける意味からも、宣言文を追加しておきます。プログラムの前のほう、変数型宣言文よりうしろに入れます。

```
25 DIM A!(5),B(5),C#(5)
```

このように、複数の配列をコンマで区切って同時に宣言することができます。

カウンタ

　次に、各配列内にデータがたまるまで、５回同じことをくりかえすわけですから、この回数を数える変数(カウンタ)を用意します。変数名は現在使われていないものでしたらＰでもＱでも何でもよいのですが、わかりやすいようにＣＯＵＮＴとします。

　この変数を使って回数をカウントしますが、これをどこに入れるかといいますと、単価入力のときに０を入力しなければまだ終りではないわけですから、このことを判断したあとでカウントするようにします。つまり、行番号40の次に入れればよいでしょう。

```
45 COUNT=COUNT+1
```

代入文

　この式がまた理解に苦しむところです。これは数学で使う等式ではありません。代入文といいます。

　なぜ代入文というかといいますと、右辺の結果、つまりＣＯＵＮＴという変数の中味に１を加えたものを、左辺のＣＯＵＮＴに代入（しまいこむ）するからです。

　たとえば、このプログラムの場合ですと、ＲＵＮコマンドによって実行したときは、すべての変数は０にクリアされますから、最初はこのＣＯＵＮＴという変数の中味も０です。そして行番号45に処理が移りますと、

右辺＝ＣＯＵＮＴ＋１＝０＋１＝１

ですから、この１という値が左辺のＣＯＵＮＴに入り、ＣＯＵＮＴの中味は１となります。つまり、最初は中味が０だったのが１になったのですから、１だけカウントされたことになります。

　そして２回目に行番号45にきますと、ＣＯＵＮＴの中味はすでに１ですから、

右辺＝ＣＯＵＮＴ＋１＝１＋１＝２

となり、２が左辺のＣＯＵＮＴに入ります。つまり、前に１だったものが２になり、また１つカウントされました。

　このような動作をくりかえしてカウントしていきますが、一般的にいうと、前の値に１だけ加算したものがＣＯＵＮＴという変数の新しい内容になるわけです。

　この「１」を５とか10にすれば、５あるいは10ずつ加算するということになります。

　この代入文は合計を算出したり、ゲームの得点計算でよく使われますし、大切な事項ですから、よく覚えておいてください。

さて、カウンタが決まりましたから、これを使って、配列に数値をしまいこむステートメントを作ります。

単価、数量、金額の値が決まったところで、これらの値を配列に代入します。ですから行番号60の次、サブルーチンへ飛ぶ行番号65より前になりますね。行番号は61～64のどれでもかまいませんが、とりあえず62としておきます。

```
62 A!(COUNT)=X!:B(COUNT)=Y:C#(COUNT)=Z#
```

1回目はＣＯＵＮＴ＝1ですからＡ！(ＣＯＵＮＴ)はＡ！(1)と同じことになり、配列Ａ！の1番目の要素に1回目の単価が入ります。2回目はＣＯＵＮＴ＝2ですから、配列Ａ！の2番目の要素に2回目の単価が入ります。配列ＢおよびＣ＃についても同じ理屈です。

●Ａ！(1)の読み方ですが、「！」をビックリマークなどと呼ぶ人をいますので「エー・ビックリ・カッコ・イチ」と読むとわかりやすいですね。

行番号62のステートメントにより、データが配列にしまわれますが、その次に行番号65のＧＯＳＵＢ文がありますから、このままでは1回目の値が格納されただけでサブルーチンに飛んでしまいます。5回分だけためたいわけです。どうしたらよいでしょうか？

ここでふたたび条件文を使います。行番号65を「もしカウンタが5になったら、サブルーチンへ飛びなさい」というステートメントに変更します。緑色の部分をインサートモードでキーインすればよいでしょう。

```
65 IF COUNT=5 THEN GOSUB *SAKUHYO
```

次に、5回カウントしてサブルーチンへ飛んでからの処理を考えてみましょう。行番号295から420までの作表部分はこのままでよいでしょう。変更するのは数値表示部分です。リストを画面に表示してください。

```
list 430-460
430 LOCATE 7,7:PRINT X!
440 LOCATE 16,7:PRINT Y
450 LOCATE 25,7:PRINT Z#
460 RETURN
Ok
█
```

この部分では１回目のデータだけ表示することができました。私たちは５回分を書き込みたいわけですね。同じことを５回くりかえせばよいのです。先ほど学んだＦＯＲ～ＮＥＸＴループを使ってみましょう。

行番号430～450を手直しして、これを５回くりかえすようにするのですから、まずＦＯＲ～ＮＥＸＴではさんでしまいましょう。次のようにキーインします。

```
425 FOR I=1 TO 5
455 NEXT I
```

80ページの図を見てください。どの行も、単価、数量、金額の表示位置のＸ座標はそれぞれ７、16、25で同じです。ところがＹ座標はどうでしょうか。下へ行くにつれて２ずつふえています。

ここでループ変数Ｉを使って、Ｙ座標を決めるようにします。計算式は次のようにします。

Ｙ座標 ➡ ７＋２×（Ｉ－１）

これは、７を初項とし、等差２の数列を算出する式です。

Ｉは１から５まで変化するわけですから、Ｙ座標が実際にどうなるか計算してみましょう。

Ｉ＝１のとき ➡ ７＋２×（１－１）＝７＋２×０＝７
Ｉ＝２のとき ➡ ７＋２×（２－１）＝７＋２×１＝９
Ｉ＝３のとき ➡ ７＋２×（３－１）＝７＋２×２＝11
Ｉ＝４のとき ➡ ７＋２×（４－１）＝７＋２×３＝13
Ｉ＝５のとき ➡ ７＋２×（５－１）＝７＋２×４＝15

この結果と80ページの図を見くらべてください。確かにあっていますね。

さあ、Ｘ座標、Ｙ座標が決まりました。行番号430から450のステートメントを次のように変更してください。緑色の部分が修正点です。これまでに学んだスクリーンエディトでチャレンジしてみてください。いやだという人は、そっくりそのままキーインしてもかまいません。

```
430 LOCATE 7,7+2*(I-1):PRINT A!(I)
440 LOCATE 16,7+2*(I-1):PRINT B(I)
450 LOCATE 25,7+2*(I-1):PRINT C#(I)
```

かけ算記号はアスタリスクを使うのでしたね。

配列はＩの値が１、２、３……と変化するに従ってＡ！（１）、Ａ！（２）、

Ａ！(３)……のようになりますから、ＬＯＣＡＴＥ文で指定されたところに表示するようにすればよいですね。

プログラムの実行

ここで、果たしてうまくいくかどうか実行してみましょう。

(f・5) キーを押してＲＵＮさせます。

```
タンカ?  400
スウリョウ? 20
```

単価を入力し数量を入力すると、ＩＮＫＥＹ＄によるキー入力待ちになります。金額が表示されないのはちょっとさびしいですね。あとでステートメントを追加しましょう。

何かキーを押すと次の入力場面です。同様の操作を５回までくりかえすと、次のように表が画面にでます。

タンカ	スウリョウ	キンガ゛ク
400	20	8000
450	3	1350
780	62	48360
150	30	4500
300	20	6000

やはり、ＩＮＫＥＹ＄によるキー入力待ちです。

次に進みたければ、どれかキーを押してください。ふたたび単価、数量の入力になります。

もう一度やってみましょうか。適当な数値を入力してください。数量の入力が終ったところで、次のようにエラーメッセージが表示され

てストップしてしまいました。

```
タンカ? 500
スウリョウ? 10
Subscript out of range in 62
Ok
■
```

83ページと同じエラーメッセージです。

これは、配列の添字がＤＩＭ文で宣言したより大きくなってしまったということです。

行番号62を見てみましょう。配列の添字にＣＯＵＮＴという変数を使っていますね。ダイレクトモードで変数ＣＯＵＮＴに何が入っているか調べてみましょう。

```
? COUNT
 6
Ok
■
```

おや？　６が入っていますね。２回目も１からカウントしなければならないのに、６になっているのです。

これは、５回カウントしてサブルーチンへ飛び、作表処理がすんだ時点でカウンタがクリアされなかったからです。カセットテープなどでも新しく曲を録音するときにカウンタをゼロにしますね。原理はまったくこれと同じです。

ＲＥＴＵＲＮ文の前にカウンタクリアのステートメントを追加しましょう。

```
457 COUNT=0
```

変数ＣＯＵＮＴの中味を０にすればクリアされたことになります。

それから、もう１つ追加ステートメントがあります。金額表示でしたね。

```
61 PRINT Z#
```

さあ、これで実行してみてください。

今度はうまくいくと思いますよ。

3・14 ほかに問題点は?

これで一応、入力データを表に書き込むことができました。ほかに問題点はないでしょうか？　一緒に考えてみましょう。

これまで、ずいぶんこま切れにしてステートメントを入力してきましたので、総リストを示すことにします。

よくコンピュータの入門書でも、行番号が10きざみにきれいにそろったものが見うけられますが、プログラミングの過程ではこのように、エラーメッセージがでては修正し、あるいは、こんな処理もしてみたいということで新しいステートメントを追加し、といったくりかえしをしながら完成させていくのです。10行くらいの短かいプログラムなら別ですが、最初から行番号のととのったプログラムなど作れません。みなさんもこの事実をよく心得て、根気よく努力してください。

話が横道にそれましたが、次が総リストです。

```
10 CONSOLE 0,25,0,0:WIDTH 40,25
20 DEFINT A-Z
25 DIM A!(5),B(5),C#(5)
30 INPUT "ﾀﾝｶ";X!
40 IF X!=0 THEN GOTO 120
45 COUNT=COUNT+1
50 INPUT "ｽｳﾘｮｳ";Y
60 Z#=X!*Y
61 PRINT Z#
62 A!(COUNT)=X!:B(COUNT)=Y:C#(COUNT)=Z#
65 IF COUNT=5 THEN GOSUB *SAKUHYO
80 Y$=INKEY$
90 IF Y$="" THEN GOTO 80
100 CLS
110 GOTO 30
120 END
290 *SAKUHYO
292 CLS
295 LOCATE 6,4
300 PRINT "┌────────┬────────┬────────┐
305 LOCATE 6,5
310 PRINT "│  ﾀﾝｶ    │ ｽｳﾘｮｳ  │ ｷﾝｶﾞｸ  │
315 LOCATE 6,6
320 PRINT "├────────┼────────┼────────┤
325 LOCATE 6,7
```

```
330 PRINT "│         │         │         │
335 LOCATE 6,8
340 PRINT "├─────────┼─────────┼─────────┤
345 LOCATE 6,9
350 PRINT "│         │         │         │
355 LOCATE 6,10
360 PRINT "├─────────┼─────────┼─────────┤
365 LOCATE 6,11
370 PRINT "│         │         │         │
375 LOCATE 6,12
380 PRINT "├─────────┼─────────┼─────────┤
385 LOCATE 6,13
390 PRINT "│         │         │         │
395 LOCATE 6,14
400 PRINT "├─────────┼─────────┼─────────┤
405 LOCATE 6,15
410 PRINT "│         │         │         │
415 LOCATE 6,16
420 PRINT "└─────────┴─────────┴─────────┘
425 FOR I=1 TO 5
430 LOCATE 7,7+2*(I-1):PRINT A!(I)
440 LOCATE 16,7+2*(I-1):PRINT B(I)
450 LOCATE 25,7+2*(I-1):PRINT C#(I)
455 NEXT I
457 COUNT=0
460 RETURN
```

処理終了の判断は正常か？

このプログラムの処理終了は、行番号40の条件文で行っています。つまり、単価として０を入力した場合、行番号 120 のＥＮＤ文に飛ばしています。

しかし考えてみてください。いつも５組のデータ、10組のデータというように、５で割り切れる区切りのよい場合ばかりだったらよいですが、７組とか13組とかのときはどうでしょうか。

７組のデータを例にとりますと、５組のデータについてサブルーチンで表に表示し、次に６組目、７組目のデータを入力して、８組目はないですから、ここで「タンカ？」と聞いてきたときに０を入力することになります。Ｘ！＝０になりましたからＥＮＤ文に飛び、Ok が表示されて処理が終了してしまいます。つまり、６組目、７組目のデ

ータ表示がないわけです。

そこで、「もしＸ！＝０で、かつカウンタが０ならば（表示すべきデータはないわけですから）ＥＮＤ文に飛ばす」ようにします。行番号40に次の緑色の部分を挿入します。

```
40 IF X!=0 AND COUNT=0 THEN GOTO 120
```

ＡＮＤは論理積といわれ、その左側と右側の式がともに真の場合に真となります。この例ですとＸ！＝０とＣＯＵＮＴ＝０がともに満たされれば、ＴＨＥＮ以下のステートメントが実行されます。 ＡＮＤ

それでは、Ｘ！＝０でカウンタが０でない場合はどうしたらよいでしょうか。この場合は表示すべきデータがあるわけですから、作表サブルーチンに飛ばさなければなりません。そして、サブルーチンから帰ってきたら終了にしてしまってよいのですから、コンピュータがその判断基準にする目じるしを作っておきます。この目じるしのことを**フラグ**といいます。 フラグ

フラグもやはり変数です。何を使ってもよいのですが、ここではわかりやすいようにＯＷＡＲＩという変数を使います。

次の行を追加します。

```
42 IF X!=0 AND COUNT<>0 THEN OWARI=1:GOS
UB *SAKUHYO
```

（<> ← 数学の不等号に相当します）

「等しくない」というときには数学で使う不等号「≠」がありませんから「＜＞」を使います。

フラグ用の変数ＯＷＡＲＩの内容を１にすることをフラグを立てるといいます。ほかではＯＷＡＲＩという変数は使っていませんから、このステートメントへくるまではいつも初期状態のままで値は０です。

このフラグによって、次にどうするかが決定されます。サブルーチンから帰ってきたあとに次のステートメントを追加します。

```
43 IF OWARI=1 THEN GOTO 120
```

フラグが立っていればＥＮＤ文へ飛べ、ということになります。

配列をクリアする

さあ、それでは７組のデータを入力してみてください。８組目のタンカの入力に対して０をキーインしてみます。すると今度は、すぐ終了しないで表が表示されました。

ところが、また１つ問題点がでてきました。８つ目、９つ目、10個目のデータも表示されているのです。入力したおぼえはないのにどうしてでしょうか。

これは、８つ目、９つ目、10個目のデータではなく、３つ目、４つ目、５つ目のデータなのです。つまり、６つ目、７つ目のデータは配列の１番目、２番目にしまわれましたが、３番目から５番目までにはまだ前のデータが残っていたのです。

ですから、サブルーチンに飛んでデータを表示しおわったら、配列内を０にクリアしておく必要があるのです。

クリアするにはＦＯＲ～ＮＥＸＴ文を使います。85ページで学習しましたね。次のステートメントを追加します。

```
458 FOR I=1 TO 5
459  A!(I)=0:B(I)=0:C#(I)=0
460 NEXT I
```

行番号460にＮＥＸＴ文を入れましたので、前のＲＥＴＵＲＮ文は消えてしまいました。行番号470としてＲＥＴＵＲＮ文を作ります。

```
470 RETURN
```

数値の桁ぞろえをする

ＰＲＩＮＴ ＵＳＩＮＧ

91ページの表示結果を見るとわかりますように、数値がすべて左づめで表示されています。これを右づめにすると大変体裁がよくなります。この要求を満たしてくれるのがＰＲＩＮＴ ＵＳＩＮＧ文です。

よく使われるのが次の３つです。

```
❶ PRINT USING "#####";A          ←数値を右づめで出力する
❷ PRINT USING "#####,";A         ←3桁ごとにコンマで区切る
❸ PRINT USING "&          &";A$  ←文字を左づめで出力する
```

❶は数値を出力する桁数を指定するものです。数値が指定された桁数（例では５桁）より小さいときは右づめで出力されます。

❷は３桁ごとにコンマが入って出力されるため、大きな数値を扱うときに見やすくなります。

❸は左の＆（アンド）マークから右の＆マークまでの字数で（例では11文字）文字列を左づめで出力します。この字数より長ければ余分

なものは無視されます。

ダイレクトモードでいくつかためしてみてください。

```
A=1234:B!=123456:A$="PC-8801"
Ok
print using "#####";a
 1234
Ok
print using "######,";b!
123,456
Ok
print using "    &              &";a$
    PC-8801
Ok
■
```

（注：4スペースあいています）

（注：スペースが4つ入り、5文字目から左づめで出力される）

PRINT　USING文にはまだいろいろな使い方がありますが、普通はこの程度で間に合ってしまいます。

行番号430から450を次のように修正してみましょう。緑色の部分が追加挿入個所です。インサートモードを使えばいいですね。

```
430 LOCATE 7,7+2*(I-1):PRINT USING "########";A!(I)
440 LOCATE 16,7+2*(I-1):PRINT USING "#########";B(I)
450 LOCATE 25,7+2*(I-1):PRINT USING "#########";C#(I)
```

81ページの欄外にコメントを入れておきましたが、このPRINT USING文を使うことによって、金額が100万以上になっても、右端のケイ線が切れるのが防げます。

小さな修正

行番号30に、処理終了の場合の指示文をいれておきましょう。

```
30 INPUT "ﾀﾝｶ (ｵﾜﾘ=0) ";X!
```

次に行番号61の金額表示にも「キンガク」と見出しをつけましょう。

```
61 PRINT :PRINT "  ｷﾝｶﾞｸ==> ";Z#
```

（注：空白行を1行出力します）

3・15 プログラムの完成

いよいよおおづめにきました。プログラムの完成も間近です。

93ページから94ページにかけてのリストを見ますと、まだ雑然としています。もう少し見やすく整理しておきましょう。

レム文を入れる

プログラムリストを見て、何のプログラムだかわかるように先頭にファイル名を入れておきます。それからプログラムの作成年月日なども入れておくとよいですね。

次のステートメントをキーインしてください。

```
1 ' ﾉｳﾋﾝｼｮ ｻｸｾｲ
2 ' file name ==> NOHIN
3 ' 1983.11.1 ｻｸｾｲ
4 ' save "NOHIN"
5 '
```

カセット使用の方はプログラム名を"CAS1：NOHIN"としてください。

各行番号のうしろにアポストロフィ（'）が入っています。これはレム文またはリマーク文といわれるもので、コンピュータは、これらの行を無視して処理は何もしません。ですから、プログラムリストを見やすくしたり、注釈を入れたりするときに使われます。

REM

アポストロフィの代わりにREMを使っても同じです。たとえば行番号1は次のようにも表現できます。

```
1 REM ﾉｳﾋﾝｼｮ ｻｸｾｲ
```

行番号4のステートメントですが、これは、このプログラムをセーブするときに使います。まずLISTコマンドで行番号4を画面に表示します。そして、カーソルを移動して「S」の上に重ねます。

```
list 4
4 ' save "NOHIN"
```

ここにカーソルを重ねる

次に (DEL) キーを4回たたきます。すると、「S」が2行目の行頭に移動しますね。

ここで、ディスク使用の方はリターンキー、カセット使用の方は録音状態にしてからリターンキーを押しますと、このプログラムがセー

ブできます。

みなさん各自ためしてみてください。

こうしておきますと、プログラムのセーブも楽ですし、ファイル名を間違える心配もありません。

行番号５のレム文は、次の行番号10からのメイン部分との区切りの意味で入れてあります。

同様に、280行、422行もレム文として次のようにキーインします。

```
280 '
422 '
```

行番号を整える

次に行番号がふぞろいなのでＲＥＮＵＭコマンドで整理しておきましょう。

```
RENUM 100
```

さあ、これですべて終りました。（f・5）キーで実行してみてください。今度はきれいに桁ぞろえもできました。次章へ進む前に、このプログラムをセーブするのをお忘れなく。



ﾀﾝｶ	ｽｳﾘｮｳ	ｷﾝｶﾞｸ
400	20	8000
450	3	1350
800	200	160000
2000	600	1200000
300	20	6000

プログラムリスト

〈注〉行番号130のＲＥＭ文はディスク内蔵タイプの場合です。カセットの場合は「ＣＡＳ１：ＮＯＨＩＮ」としてください。次章からも同じ扱いになりますから注意してください。

3·1節からスタートした、3行からなるプログラムがずいぶん長いものになってしまいました。しかし、プログラム作成というのは、このように小さなものから、だんだん大きくしていくのが筋道です。この辺の感じをつかんでいただければ幸いです。

次に、このプログラムの総リストを示します。

```
100 ' ﾉｳﾋﾝｼｮ ｻｸｾｲ                          ┐
110 ' file name ==> NOHIN                  ├─注釈
120 ' 1983.11.1 ｻｸｾｲ                       │
130 ' save "NOHIN"                         ┘
140 '
150 CONSOLE 0,25,0,0:WIDTH 40,25 ←── 画面初期設定
160 DEFINT A-Z ←── 変数の整数宣言
170 DIM A!(5),B(5),C#(5) ←── 配列の宣言
180 INPUT "ﾀﾝｶ (ｵﾜﾘ=0) ";X! ←── 単価入力
190 IF X!=0 AND COUNT=0 THEN GOTO 320            ┐
200 IF X!=0 AND COUNT<>0 THEN OWARI=1:GOS        ┘
UB *SAKUHYO                                      終了処理
210 IF OWARI=1 THEN GOTO 320
220 COUNT=COUNT+1 ←── カウンタ
230 INPUT "ｽｳﾘｮｳ";Y ←── 数量入力
240 Z#=X!*Y ←── 金額計算
250 PRINT :PRINT "  ｷﾝｶﾞｸ==> ";Z#
260 A!(COUNT)=X!:B(COUNT)=Y:C#(COUNT)=Z# ←── データを配列に格納
270 IF COUNT=5 THEN GOSUB *SAKUHYO
280 Y$=INKEY$                        ┐
290 IF Y$="" THEN GOTO 280           ┘── キー入力待ち
300 CLS ←── 画面クリア
310 GOTO 180
320 END
330 '
340 *SAKUHYO
350 CLS
360 LOCATE 6,4
370 PRINT "┌─────────┬─────────┬─────────┐
380 LOCATE 6,5
390 PRINT "│  ﾀﾝｶ    │ ｽｳﾘｮｳ   │ ｷﾝｶﾞｸ   │
400 LOCATE 6,6
410 PRINT "├─────────┼─────────┼─────────┤
420 LOCATE 6,7
```

```
430 PRINT "│          │          │          │
440 LOCATE 6,8
450 PRINT "├──────────┼──────────┼──────────┤
460 LOCATE 6,9
470 PRINT "│          │          │          │
480 LOCATE 6,10
490 PRINT "├──────────┼──────────┼──────────┤
500 LOCATE 6,11
510 PRINT "│          │          │          │
520 LOCATE 6,12
530 PRINT "├──────────┼──────────┼──────────┤
540 LOCATE 6,13
550 PRINT "│          │          │          │
560 LOCATE 6,14
570 PRINT "├──────────┼──────────┼──────────┤
580 LOCATE 6,15
590 PRINT "│          │          │          │
600 LOCATE 6,16
610 PRINT "└──────────┴──────────┴──────────┘
620 '
630 FOR I=1 TO 5
640 LOCATE 7,7+2*(I-1):PRINT USING "####
####";A!(I)
650 LOCATE 16,7+2*(I-1):PRINT USING "###
#####";B(I)
660 LOCATE 25,7+2*(I-1):PRINT USING "###
#####";C#(I)
670 NEXT I
680 COUNT=0
690 FOR I=1 TO 5
700   A!(I)=0:B(I)=0:C#(I)=0
710 NEXT I
720 RETURN
```

数値表示（640〜660）

カウンタをリセット（680）

配列をクリア（690〜710）

結果をプリンタへ出力するには

結果を画面に表示しないでプリンタで印刷したい方は、行番号 370 から 660 までの「ＰＲＩＮＴ」をすべて「ＬＰＲＩＮＴ」に変更してください。ＰＲＩＮＴの前に「Ｌ」を１字挿入するだけでよいのです。

ＬＰＲＩＮＴ

ディスク内蔵タイプをご使用の方へ

私たちは1・3節で、新しいディスクを2枚用意し、次の2つのディスクを作りました。

❶ **オリジナルのシステムディスクのコピー**
❷ **システムだけをコピーしたディスク**

そして私は、❷のディスクはオリジナルのシステムディスクとともに安全な場所に保管しておくように申し上げましたので、あなたは現在❶のディスクをご使用と思います。

しかし、❶のディスクのフリーエリアをテストしてみますと、デモプログラムのデータがぎっしりつまっているため、私たちの使える部分があまりありません。このままで進みますと途中で、

```
Disk full
```

のエラーメッセージが出そうです。

そこで恐縮ですが、❶のディスクをドライブ1に入れ、❷のディスクを用意してください。そして次のようにキーインしてください。

```
load "dskut2.n88" ⏎
```

（"dskut2" の数字「2」を指して）モデル20の人はこの数字を「1」にしてください

「Ok」が表示されたらドライブ1から❶のディスクを取り出し❷のディスクをドライブ1に入れてプログラムをセーブします。

```
save "dskut2.n88" ⏎
```

〈注〉セーブ、ロードは2つのドライブを使えばディスクを入れ換えないでよいのですが、モデル20の方もいますのでこのようにしました。

同じ操作により❶のディスクの「@ load. n88」を❷のディスクにセーブしてください。次にもう一度❶のディスクをドライブ1に入れ、いまメモリ上にある「@ load. n88」をRUNさせてください。

画面に206ページのメッセージが表示されたら❷のディスクをドライブ1に入れ次のようにキーインしてください。

```
bsave "@exst",&hdc00,&ha00 ⏎
```

＊このプログラムはマシン語ですので従来の方法でセーブしようとすると
Illegal direct
のエラーになります。

これは、拡張機能を使うためのマシン語プログラムを❷のディスクにセーブしているのです。❶のディスクに「@ exst」* というプログラムはあるのですが、マシン語のためSAVEコマンドは使えません。

「Ok」が表示されたらセーブ終了です。これからは、システムとユーティリティプログラムだけが入っている❷のディスクを使用し、❶のディスクは保管しておいてください。

第4章

ブロックくずし

4・1　まず画面レイアウトから

ブロックくずしとは、下の図のように、四角いブロックをちりばめたコートを作り、ラケットを左右に操作して、はねかえってくるボールを打ちかえしブロックにあてて少しずつくずしていくゲームです。

初期設定

画面のモードとか変数の初期値などを決めておくことを**初期設定**といいます。

画面は40桁×25行のカラーモードを使い、ファンクションキーの表示はじゃまなので消すことにします。WIDTH文、CONSOLE文を使ってマルチステートメントにします。

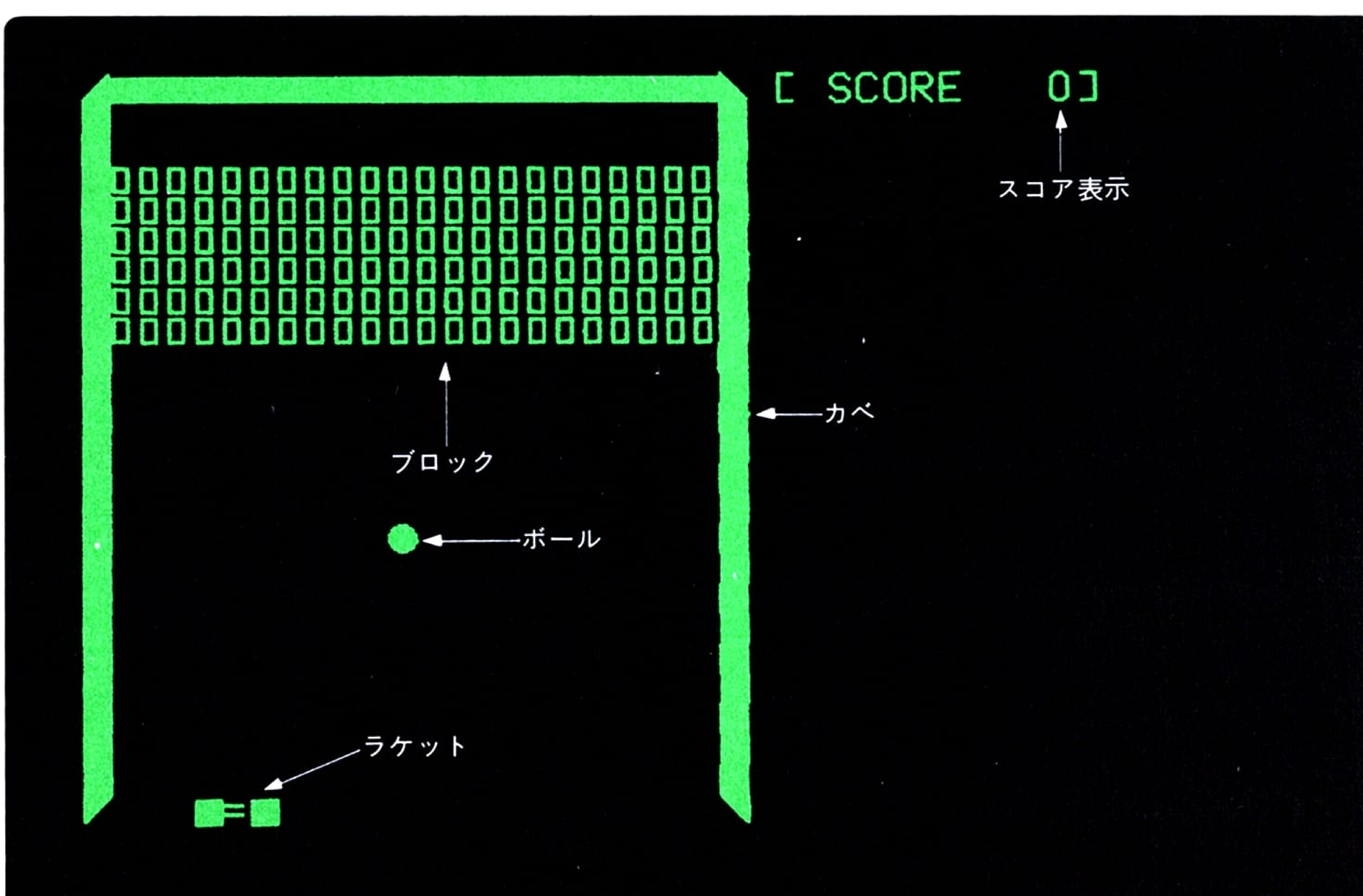
```
10 WIDTH 40,25:CONSOLE 0,25,0,1
```

ファンクションキー無表示（0）　カラーモード（1）

次に、はじめ画面を黄色にして、まわりのカベとスコアを表示することにします。カラーコードは39ページを参照してください。

```
20 COLOR 6
```

カラーを黄色にする



これはゲームですから、スピード感があったほうがいいですね。35ページで説明したように、すべての変数の整数宣言をします。

そして、私たちがいま見ているディスプレイのホームポジションの位置にコンピュータ内では番地がつけられていて、これが16進数のＦ３Ｃ８番地になっています。コンピュータ内のディスプレイ用に割り当てられた記憶領域を**ＶＲＡＭ（ブイラム、またはビデオラム）**といいますが、この先頭番地がＦ３Ｃ８になっているわけです。あまり理屈を考えないで覚えてしまってください。

このゲームが進行しますと、ボールがあちこち飛びまわりますので、このボールがいまどこにあるか、またブロックに当たったかどうかなどの判断が必要になります。そのときの基準になるのが、ＶＲＡＭの先頭番地です。ですから、これを初期値として変数にしまっておきましょう。

変数名には、わかりやすいように「ＶＲＡＭ」を使いましょう。

次のステートメントをキーインします。

ＶＲＡＭ
●コンピュータ内の記憶領域は、コマンド・ステートメントをしまっておくところ、プログラムを覚えておくところ、配列を格納するところなど、その役割によって場所が決められています。そしてその場所には私たちの住所と同じように番地（アドレス）がつけられていてテレビ画面の記憶を受けもつビデオラムはＦ３Ｃ８番地からになっています。

```
30 DEFINT A-Z:VRAM=&HF3C8
```

アンドエイチをつける

Ｆ３Ｃ８の前に「＆Ｈ」をつけてください。なぜかといいますと、左辺の「ＶＲＡＭ」は整数用の変数ですね。一方、Ｆ３Ｃ８というのは数値ではなく文字列（ストリング）です。ですから、この16進数の文字列を10進数に変換しておかなければ正しく代入されません。この変換するための記号が「＆Ｈ」というわけです。

ダイレクトモードで次のように打ち込んでみてください。

```
? F3C8
 0
Ok
? &HF3C8
-3128
Ok
█
```

最初は「＆Ｈ」をつけませんでした。すると、その結果は０と表示されました。つまり、Ｆ３Ｃ８が一種の変数とみなされ、その中味が表示されたのです。

次に「＆Ｈ」をつけました。今度は、マイナス符号がついていますが、ちゃんとした10進数に変換されましたね。

次は画面のクリアです。これは文句なくわかりますね。

```
40 CLS
```

まわりのカベを作る

私たちは前章で、グラフィックキャラクタを使って表を画面に描き、それをＰＲＩＮＴ文に作りかえてきました。周囲にカベを作るのも原理は同じですから、同じ操作をくりかえせばよいのですが、ここでは少し違った方法でやってみます。

まず、上の天井を表示するようにしましょう。

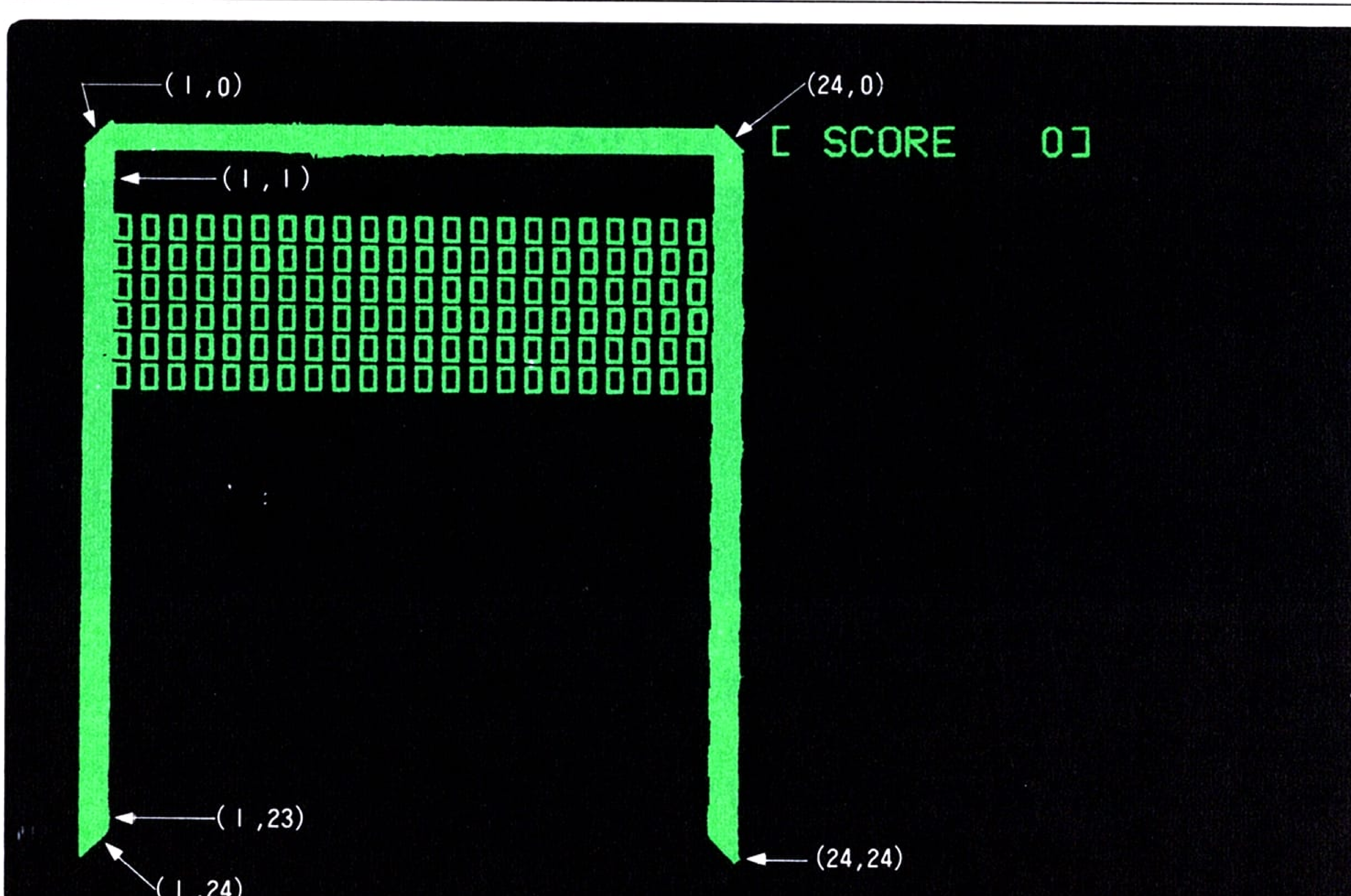
座標(１，０)のところにあるグラフィックキャラクタは GRPH キーと E キーを押したとき表示されるキャラクタです。

座標(24，０)のキャラクタは、 GRPH キーと R キーを押したときのものです。また、両はじの三角形にはさまれた部分を表示するには、 GRPH キーと M キーを押したときのキャラクタを使います。

さて、使用するキャラクタが決まりました。次のようにキーインすればよいですね。行番号を50にします。（これはまだ入力しないでください）

```
50 PRINT "                              "
```





これでも確かに間違いなく表示されます。しかし、ここで１つ新しい知識を吸収しましょう。**ＳＴＲＩＮＧ＄（ストリングダラー）関数**を使います。あるキャラクタを一定個数出力するものです。

ＳＴＲＩＮＧ＄関数

ＳＴＲＩＮＧ＄関数の書式は次の通りです。

```
STRING$(22,"■")
STRING$(22,&H87)
```


この例は２つとも同じ結果を出力します。

文字指定の場合には、ダブルクォーテーションでくくってください。また、キャラクタは１個しか有効ではありませんから、３つ入れたとしても最初のキャラクタきり表示されません。

数値指定の場合には、希望のキャラクタを下の表でさがし、横位置、縦位置の順で座標を決め、その前に「＆Ｈ」をつけます。つまり「＆

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E		0	@	P		p				ー	タ	ミ		
1	S_H	D_1	!	1	A	Q	a	q			。	ア	チ	ム		円
2	S_X	D_2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	E_X	D_3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	E_T	D_4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	E_Q	N_K	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	A_K	S_N	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	B_L	E_B	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	B_S	C_N	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	
9	H_T	E_M	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	
A	L_F	S_B	*	:	J	Z	j	z			ェ	コ	ハ	レ	♦	
B	H_M	E_C	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	
C	C_L	→	,	<	L	¥	l	¦			ャ	シ	フ	ワ	●	
D	C_R	←	ー	=	M	]	m	}			ュ	ス	ヘ	ン	○	
E	S_O	↑	.	>	N	^	n	~			ョ	セ	ホ	゛		
F	S_I	↓	/	?	O	_	o				ッ	ソ	マ	゜		

キャラクタコード

Ｈ８７」が「■」の**キャラクタコード**になっているのです。

この例では、いずれも「■」を22個表示します。

それでは、ＳＴＲＩＮＧ＄関数を使って天井を表示するステートメントを作りましょう。キャラクタコードは例とは違って＆Ｈ８６です。

```
50 PRINT " ◢";STRING$(22,"■");
60 PRINT "◣[ SCORE    0]"
```

← セミコロンをつけてください

ＣＨＲ＄について

●これはキャラクタドルと呼ばれる関数で、前ページの表のキャラクタコードを指定すると、対応した文字が与えられます。たとえば

Ａ＄＝ＣＨＲ＄(＆Ｈ87)

とすれば、Ａ＄には「■」が格納されます。この関数を使って周囲のカベを作ることもできますが、混乱するので割愛します。

行番号50の一番うしろのセミコロンを忘れないでください。これがないと、行番号60のスコア表示は次の行に改行されてしまいます。

次は、左右のカベです。まず左のカベから……。

106ページの図を見ていただくとわかりますが、座標(１，１)から座標（１，23）まで黒い四角を出力してつなげています。このキャラクタは、GRPH キーと ネ のキーを押したときに表示されるものです。全部で23個です。そしていずれの表示位置も、Ｘ座標は一定で、Ｙ座標だけが１ずつふえています。

前章で学んだＦＯＲ～ＮＥＸＴ文とＬＯＣＡＴＥ文の組み合せで作れますね。次のようになります。

```
70 FOR I=1 TO 23
80   LOCATE 1,I:PRINT "■"
90 NEXT I
```

ループ変数Ｉを１ずつ変化させれば、ＬＯＣＡＴＥ文によって表示位置が１つずつ下がっていきますね。そこにＰＲＩＮＴ文で黒四角を出力すればよいわけです。

それでは右側のカベも続けてキーインしてしまいましょう。今度はＸ座標が24になっただけです。左側のときとまったく同じようにすればよいですね。

```
100 FOR I=1 TO 23
110   LOCATE 24,I:PRINT "■"
120 NEXT I
```

そうしましたら、左右のカベの一番下に三角形をくっつけます。それぞれのキャラクタにどのキーが対応しているか、もうわかりますね。

次のようにキーインします。

```
130 LOCATE 1,24:PRINT "◤";SPC(22);"◥"
```

スペースを22個出力します

はじめてお目にかかる関数があります。ＳＰＣ（スペース）関数です。これは、カッコ内に数字を指定すると、その指定した数だけスペースを出力してくれます。

ＳＰＣ関数

ブロックを表示する

106 ページの図を見てください。ブロックは４行目から９行目にかけて横に22個並べてあります。４行目のＹ座標は３、９行目が８ですから、ＬＯＣＡＴＥ文でＸ座標は２に固定し、Ｙ座標だけを３から８まで変化させ、そこに22個のブロックをＰＲＩＮＴ文で表示するようにします。ブロックのキャラクタにはカタカナの「ロ」を使います。

また、表示色は気分転換して紫色にします。次のようにキーインします。

```
140 COLOR 3
150 FOR I=1 TO 6
160   LOCATE 2,2+I
170   PRINT STRING$(22,"ロ")
180 NEXT I
```

さて、これで画面づくりがかたづきました。

ここでＲＵＮさせてみますが、その際、次のステートメントをキーインして実行してみてください。

```
185 Y$=INKEY$:IF Y$="" THEN 185
```

入力が終りましたら （f･5） キーを押して実行です。

画面はどうなりましたか？　次ページの画面のように天井とスコア表示が消えてしまいましたね。

これはなぜかと申しますと、画面の一番下に対するＰＲＩＮＴ文のうしろにセミコロンがないため、１行だけ上へスクロールしてしまったのです。行番130 の一番うしろにセミコロンをつけてください。

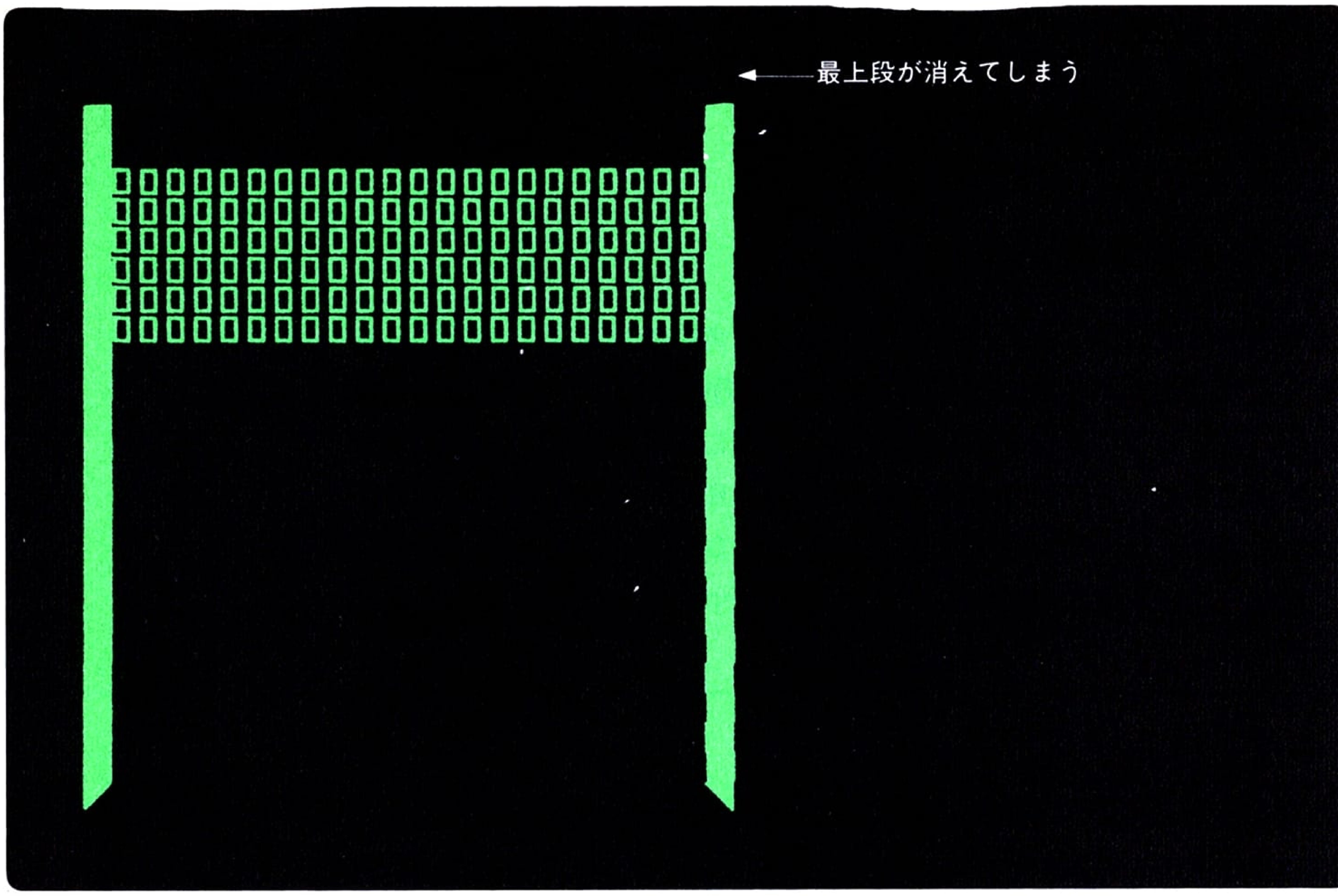
```
130 LOCATE 1,24:PRINT "◤";SPC(22);"◥";
```

セミコロンをつける

（STOP） キーでプログラムを止めてからキーインしてくださいよ。

終りましたら、もう一度ＲＵＮさせてみてください。今度はうまくいったと思います。プログラムを止めて185 行を削除しましょう。

```
185 ↵
```




```
10 WIDTH 40,25:CONSOLE 0,25,0,1
20 COLOR 6
30 DEFINT A-Z:VRAM=&HF3C8
40 CLS
50 PRINT " ◢";STRING$(22,"■");
60 PRINT "◣ [ SCORE   0]"
70 FOR I=1 TO 23
80   LOCATE 1,I:PRINT "■"
90 NEXT I
100 FOR I=1 TO 23
110   LOCATE 24,I:PRINT "■"
120 NEXT I
130 LOCATE 1,24:PRINT "◤";SPC(22);"◥";
140 COLOR 3
150 FOR I=1 TO 6
160   LOCATE 2,2+I
170   PRINT STRING$(22,"□")
180 NEXT I
```

ここまでのプログラムリストを上に示します。間違いがないか、もう一度チェックしてみてください。

4・2　変数の初期設定

この節では、ラケットとボールの座標の初期設定をします。

下の図をごらんください。ラケットは、Y座標が24の水平線上を左右に移動します。また、ボールはY座標23の水平線上のどこかから飛び出してくるようにします。そして、ラケットは初期位置を固定しておきます。左端が座標(10, 24)の位置とします。

ラケットは左右に動くだけですから、X座標だけが変化します。そして左右のカベにつきあたったら動けませんので、ぶつかったかどうかをチェックする変数ＬＩＭＩＴを設定します。この変数の初期値は下の図からもわかるように

ＬＩＭＩＴ=10

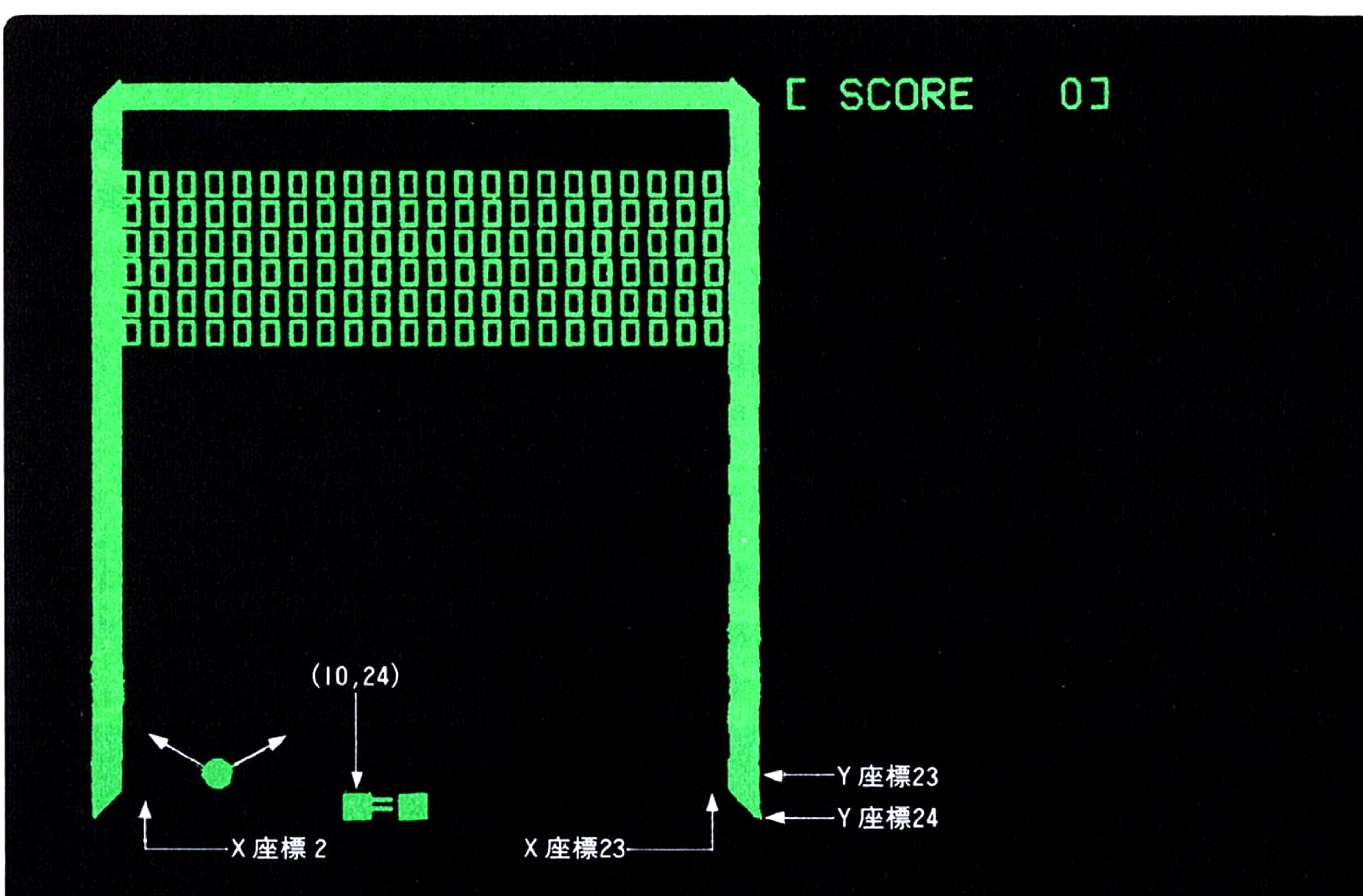
です。

次にボールのX座標、Y座標用の変数を次のように設定します。

X座標➡ＢＡＬＬ.Ｘ　　Y座標➡ＢＡＬＬ.Ｙ

ボールは最初、Y座標23の水平線上から飛び出しますから、

ＢＡＬＬ.Ｙ=23



です。

次にボールのX座標ですが、いつも不特定のところから飛びだすようにしなければなりませんから、**RND(ランダム)関数**によって決めるようにします。これは乱数を発生させる関数です。

RND関数

次のようにダイレクトモードでキーインしてみてください。

```
for i=1 to 5:r=rnd(1):? r:next i
 .245121
 .305003
 .311866
 .515163
 .0583136
Ok
█
```

いかがですか? 小数が5つ表示されましたね。このようにRND(1)は0より大きく1未満の実数をランダムに発生します。

この乱数関数を使ってボールのX座標を決定しましょう。図を見ると、ボールはX座標2から23までの間に出現すればよいわけですから、乱数関数で0から21までの数を作り出して、それに2をプラスすればよいことになります。0から21までの乱数はRND(1)に22をかければよいので、結局ボールのX座標は、次のようになります。

BALL.X=RND(1)*22+2〈注〉

〈注〉右辺が小数点つきの実数でも、左辺の変数が整数宣言してありますから、小数点以下は四捨五入されて整数が入ります。

次に、ボールは上下左右に移動します。したがって、その移動量も考慮しなければなりません。左右、上下の移動量に対する変数を

MOVE.X　MOVE.Y

としましょう。最初は下へはいけませんからMOVE.Y=−1です。

左右の移動ですが、これはどちらへ動いてもよいので、

INT(インテジャー)関数

INT関数は

INT(数式)

で表わされ、カッコ内の値の小数点以下を切り捨てた値を与えます。

MOVE.X=INT(RND(1)*2)*2−1

(INT(RND(1)*2) — 0または1の値をとる)

となります。この左辺はいつも−1または1が与えられます。つまり、いつも右か左かに移動することになります。この部分のプログラムは次のようになります。

```
190 BALL.Y=23:MOVE.Y=-1:LIMIT=10
200 BALL.X=RND(1)*22+2
210 MOVE.X=INT(RND(1)*2)*2-1
```

4・3 ラケットを動かす

このゲームでのキー操作は、ラケットを左右に動かすだけでよいので、テンキーの 1 と 3 を使うことにします。1 のキーを押せばラケットが左に動き、3 のキーを押すと右に動くようにします。

ですから、1 のキーを押したか 3 のキーを押したか、それをキャッチするステートメントを作らなければなりませんね。この条件を満たしてくれるものに、ゲームではよく使われる**ＩＮＰ（インポート）関数**があります。 ＩＮＰ関数

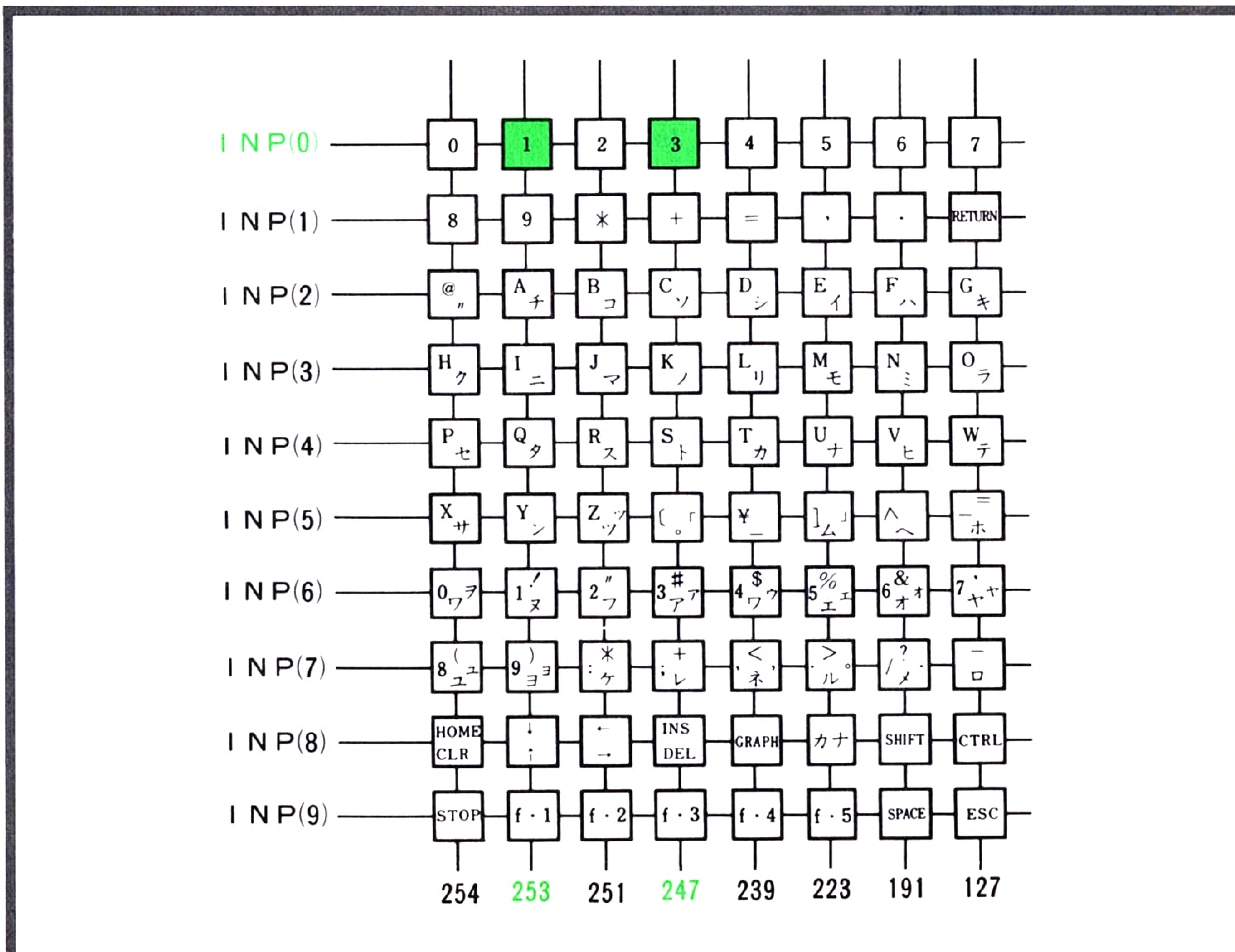
次の図をごらんください。

キーボード上のキーにはそれぞれ入力ポートがあり、そのポート番号によって、ある一定の値が対応しています。

ＩＮＰ関数の書式は

ＩＮＰ(ポート番号)

ですから、たとえばポート０を指定すると、キー 1 には253が対応し、キー 3 には247が対応します。



次のようにダイレクトモードでキーインし、[1]か[3]のキーを押してみてください。

```
for i=1 to 5000:? inp(0):next i
 255 ┐
 255 ┘──────どのキーも押していない
 253 ┐
 253 ┘──────1のキーを押した
 247 ┐
 247 │──────3のキーを押した
 247 ┘
```

どうですか？ [1]のキーを押せば253が、[3]のキーを押せば247が表示されたでしょう。

それでは、このINP関数を使ってステートメントを作りましょう。この関数の値を変数Iに代入しておきます。

```
220 '
230 *MAIN
240 I=INP(0)
```

これから作成する部分はメインとなるルーチンですので行番号230でラベルをつけておきます。

ラケットの色は水色にしますので、ここでカラーコードを5に変えておきます。

```
250 COLOR 5
```

さて、次はラケットの左右への移動を考えます。

ラケットのX座標を

LIMIT

[1] [3] キーを押すことによって左右に移動しますから、その移動量を

RACKET.X

という変数で扱うことにしましょう。とにかく、次のステートメントをキーインしてください。

```
260 RACKET.X=(I=247)*(LIMIT<21)-(I=253)*
(LIMIT>2)
```

ずいぶんおかしな式が登場しましたね。ゆっくり考えていきましょう。まず「(I=247)」について考えてみます。

これは論理式で、これが真の場合、つまり左辺の変数の中味と右辺の247を比較しそれが等しければ「－1」が与えられ、偽の場合、つまり、いまの逆の場合には「0」が与えられます。

ダイレクトモードで確認してみましょう。

```
I=247:? (I=247)
-1
Ok
I=255:? (I=247)
 0
Ok
■
```

わかりましたか？

では「(LIMIT<21)」はどうでしょうか。

これも同じ理屈です。変数LIMITの内容が21未満であれば、この論理式は「真」ですから「－1」が、逆に21以上であれば「0」が返ってきます。

```
LIMIT=2:? (LIMIT<21)
-1
Ok
LIMIT=25:? (LIMIT<21)
 0
Ok
■
```

間違いありませんでしたね。

それでは、上の2つの論理式の積

❶ (I=247)*(LIMIT<21)

の値はどうなるでしょうか？

もうわかりますね。そうです、変数Iの中味が247で、かつ変数LIMITの中味が21未満の場合に限り、「－1」と「－1」の積で「1」が与えられ、それ以外のときは常に「0」となります。

これは何を意味するかといいますと、Iが247ということは [3] のキーが押されていることです。そしてLIMIT<21ということは、ラケットの右端が右側のカベに接触していない（ラケットの長さは3

キャラクタ分であることに注意してください）ということになります。
同じように

❷ (I=253)*(LIMIT>2)

についても考えてみてください。
I＝253、つまり [1] のキーが押されていて、かつラケットの左端と左側のカベとの間に1キャラクタ分のアキがあるときにのみ、「1」になります。
ここで総括してみますと、❶が「1」のときは❷は「0」ですから

RACKET.X＝1－0＝1

となり、右へ1だけ移動します。
一方、❷が「1」のときは❶は「0」ですから

RACKET.X＝0－1＝－1

となり、左へ1だけ移動することになります。
そして、上記以外の場合、つまり [1] のキーも [3] のキーも押されていないとか、あるいは左か右のカベにぶつかってしまったときには、

RACKET.X＝0－0＝0

となって、ぜんぜん動かないことになります。

それでは、この移動量をもとにラケットを表示することにしましょう。
その手順ですが、まず移動前のラケットを消去して、次に新しい位置に表示します。ラケットの移動前の座標は

(LIMIT, 24)

ですから、この位置にＬＯＣＡＴＥ文を使って３キャラクタ分のスペースを出力します。この操作でラケットは消えてしまいます。ステートメントは次のようになります。

```
270 LOCATE LIMIT,24:PRINT "   ";
```

そして、ラケットの新しい座標を設定します。

```
280 LIMIT=LIMIT+RACKET.X
```

前のＸ座標に移動量を加え、それを変数ＬＩＭＩＴに代入すればいいですね。

あとは、この新しい座標にラケットを表示するだけです。

```
290 LOCATE LIMIT,24:PRINT "■=■";
```
└セミコロンをつけ忘れないように

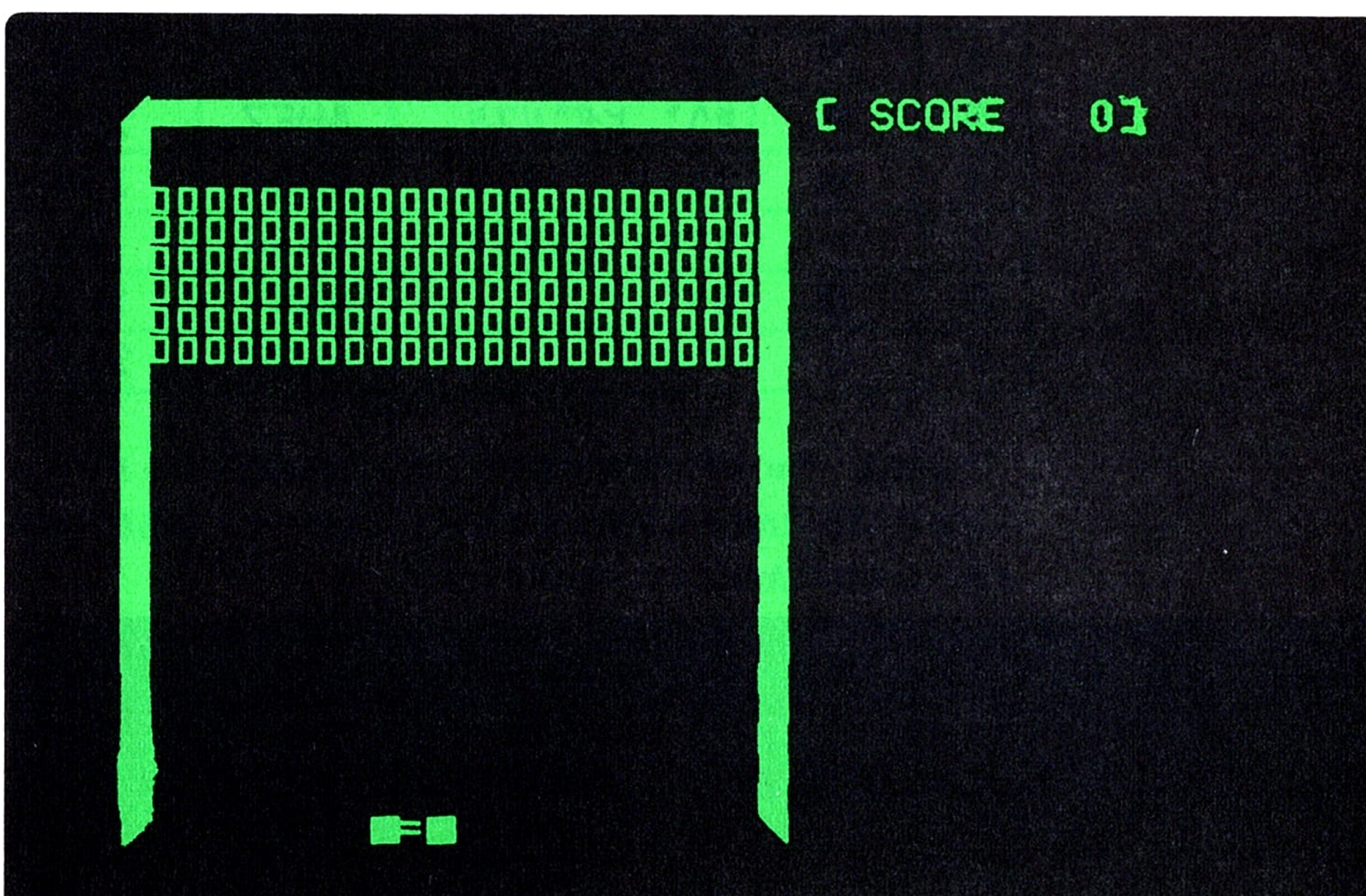
次から、ボールをセットするルーチンが入りますので、ボールの表示色の白にＣＯＬＯＲ文を追加しておきます。

```
300 COLOR 7
```

この部分のプログラムリストと実行結果を示します。

みなさんもＲＵＮさせてみてください。

```
220 '
230 *MAIN
240 I=INP(0)
250 COLOR 5
260 RACKET.X=(I=247)*(LIMIT<21)-(I=253)*
(LIMIT>2)
270 LOCATE LIMIT,24:PRINT "   ";
280 LIMIT=LIMIT+RACKET.X
290 LOCATE LIMIT,24:PRINT "■=■";
300 COLOR 7
```



4・4 ボールが何に当たった？

ラケットの移動と表示はできました。それではボールについてはどうでしょうか？

ボールは、このコートの中を常に動きまわっています。いいかえますと、ボールは常にラケットのキャラクタ、カベのキャラクタ、ブロックのキャラクタ、そして、何も見えない空間をつくっている「空白」というキャラクタなどに衝突しているのです。

一方、ボールがカベとかブロックとかラケットに当たれば、はねかえらなければいけませんし、何もなければ直進できますね。ということは、ボールが衝突したところのキャラクタが何であるかわかれば、その先のボールの移動方向が決定できるわけです。そして、このプログラムでは、どのキャラクタかを判別するのに107ページのキャラクタコード表の16進数の該当コードを10進数に変換した値を使うことにします。下の表をごらんください。

No.	キャラクタ	タイショウブツ	16シンスウ	10シンスウ
1		スペース	&H20	32
2	◢	テンジョウ ヒダリ	&HE4	228
3	◣	テンジョウ ミギ	&HE5	229
4	◥	カベ ミギスミ	&HE6	230
5	◤	カベ ヒダリスミ	&HE7	231
6	■	ラケット ノ サユウ	&H86	134
7	=	ラケット チュウオウ	&H3D	61
8	■	カベ	&H87	135
9	▯	ブロック	&HDB	219

この表は、ブロックが衝突するキャラクタの16進数をキャラクタコード表から調べ10進数に変換した値を示したものです。16進数から10進数への変換は次のようにすればできます。

```
? &h20,&he4
 32            228
Ok
█
```

画面上のある位置の内容をさぐる

ブロックが衝突するキャラクタと、その10進表記のコードが判明しましたので、この問題は一時保留して次にうつります。

画面上に、あるキャラクタがあった場合、それをどのようにして調べるかということです。これには**PEEK（ピーク）文**を使います。画面を40桁のモードにしたとき、座標（X,Y）にあるキャラクタの10進数コードは、次の公式によって求められます。

P＝PEEK（Y＊120＋X＊2＋&HF3C8）

PEEK

●「PEEK」とは米語のスラングで「のぞき見る」ということです。ブイラムの中をのぞく、ちょっとわかるような気がしませんか。

たとえば、座標（5，3）の位置に「■」を表示して、この公式を使って、その表示された位置のキャラクタコードを調べてみましょう。次のようにキーインしてみてください。

```
locate 5,3:? "■"

     ■
Ok
p=peek(3*120+5*2+&hf3c8)
Ok
? p
 135  ←黒四角のキャラクタコード
Ok
■
```

うまくいきましたね。

ですから、ボールの座標さえわかれば、その動いた位置にどんなキャラクタがあるか、たちどころにわかってしまうわけです。

それではボールの座標はどのように決めたらよいでしょうか？

そうです、次のようにすればよいですね。

X座標＝（ボールのもとのX座標）＋（移動量）
Y座標＝（ボールのもとのY座標）＋（移動量）

4・2節で、これらの座標は決めましたね。もう一度確かめてください。変数を使って表わすと、次のようになります。

X座標＝BALL.X＋MOVE.X
Y座標＝BALL.Y＋MOVE.Y

そうしますと、ボールが動いた位置にあるキャラクタのコードを調

べるには、公式を使って次のようにします。

```
P=PEEK((BALL.Y+MOVE.Y)*120
  +(BALL.X+MOVE.X)*2+VRAM)
```

さあ、ボールがぶつかったキャラクタのコードをチェックするルーチンを作りましょう。

ラケット移動ルーチンを抜けでたらすぐにチェックするようにします。そして、これはよく使いますのでサブルーチンにしておきます。まず次のステートメントです。

```
310 GOSUB *CHECK
```

そして、このサブルーチンをキーインします。

```
1000 '
1010 *CHECK
1020 P=PEEK((BALL.Y+MOVE.Y)*120+(BALL.X+
MOVE.X)*2+VRAM)
1030 RETURN
```

ボールの動く方向は?

●ここでは、いくつか発生する条件によってプログラムの処理を分岐させることになりますので、
IF～THEN文
を使います。わからない方は58ページを参照ください。

前項で、ボールが動いた位置にどんなキャラクタがあるかは、その場所をＰＥＥＫ文で調べることによってわかりましたが、それではボールは、そのキャラクタの違いによってどんな方向に飛んでいくでしょうか?　ここでは、このことを考えてみましょう。

ボールの移動するケースですが、右ページの図をごらんください。❶から❿までが考えられます。⓫はコートから出てしまったのでゲームオーバーにしてしまいます。

❶の場合

この場合は4方向どの向きにも動くことができます。そして、1つ移動しましたから、移動前のボールを消して、新しい位置にボールをセットしなければなりません。

❶の場合の判断基準は先ほどのキャラクタコードのチェックサブルーチンで調べた値がスペース、つまりＰ＝32ということになります。

Ｐ＝32だったら、ボール表示のサブルーチンへ飛ばしましょう。

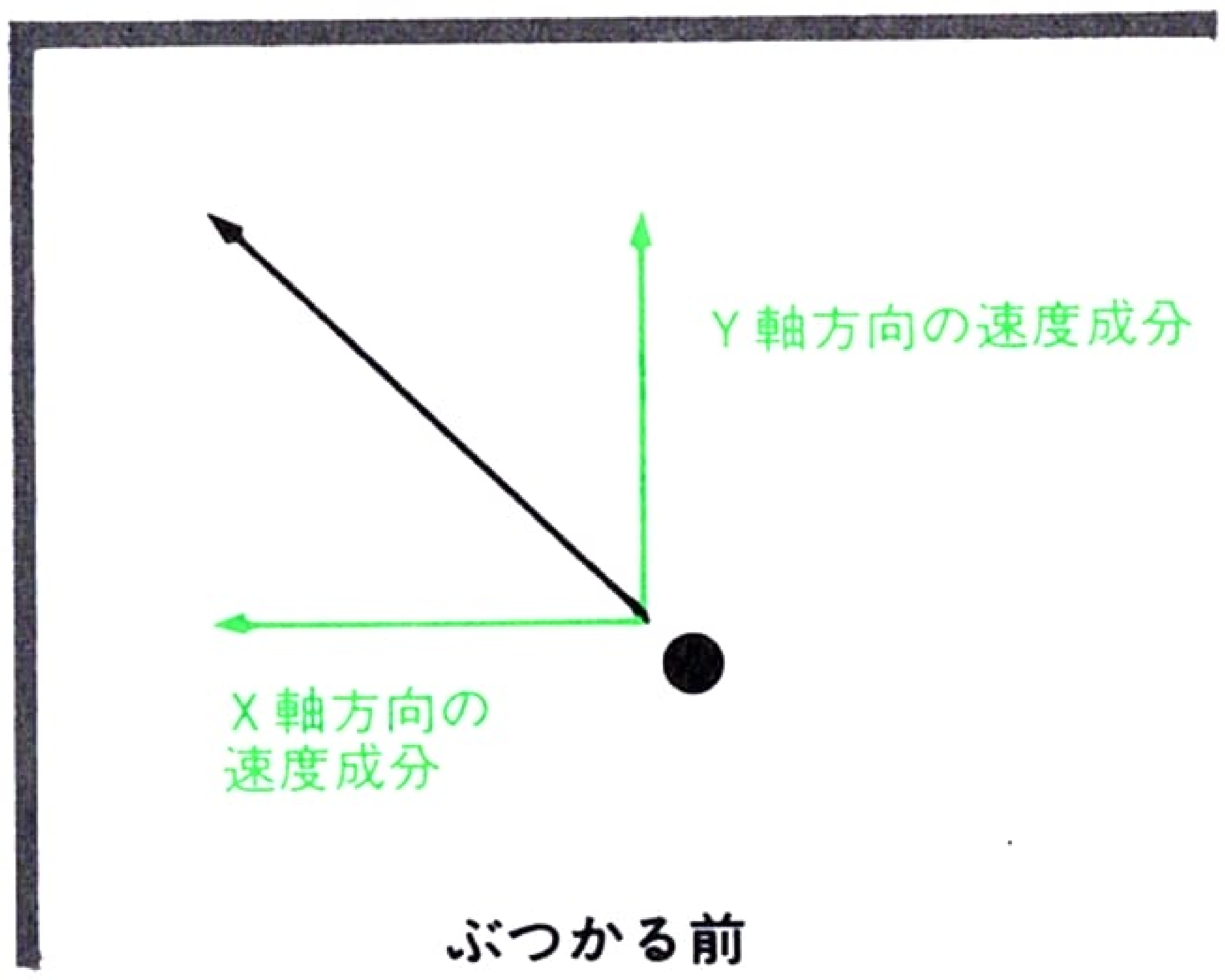
```
320 IF P=32 THEN GOTO *SET.BALL
```

ボールをセットするサブルーチンは、あとで作ります。

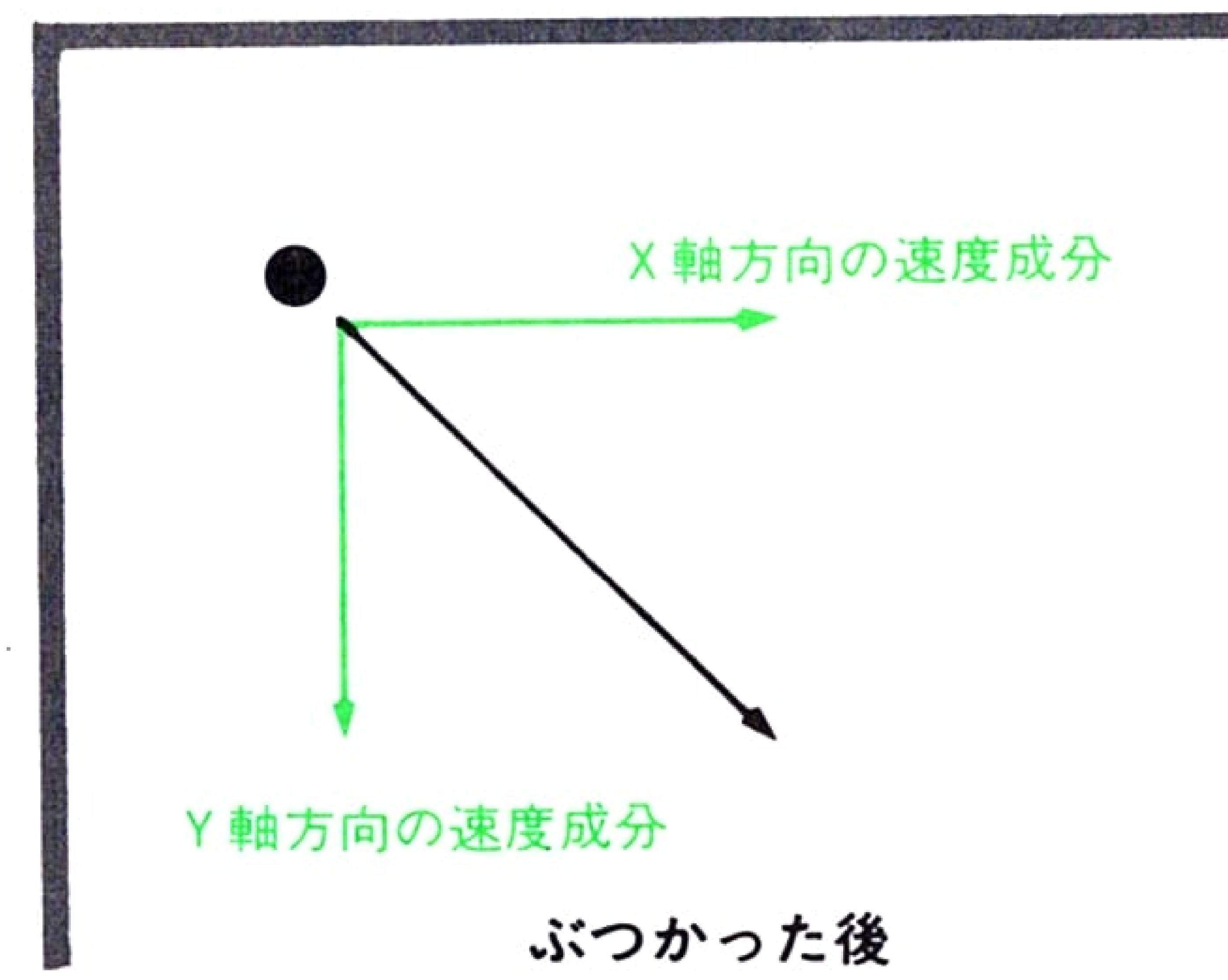
❷〜❺の場合

これは、コートの4すみにぶつかるケースです。この場合には、ぶつかる前とぶつかったあとでは進行方向が正反対になります。

たとえば❷のときを考えますと、次の図のようになります。

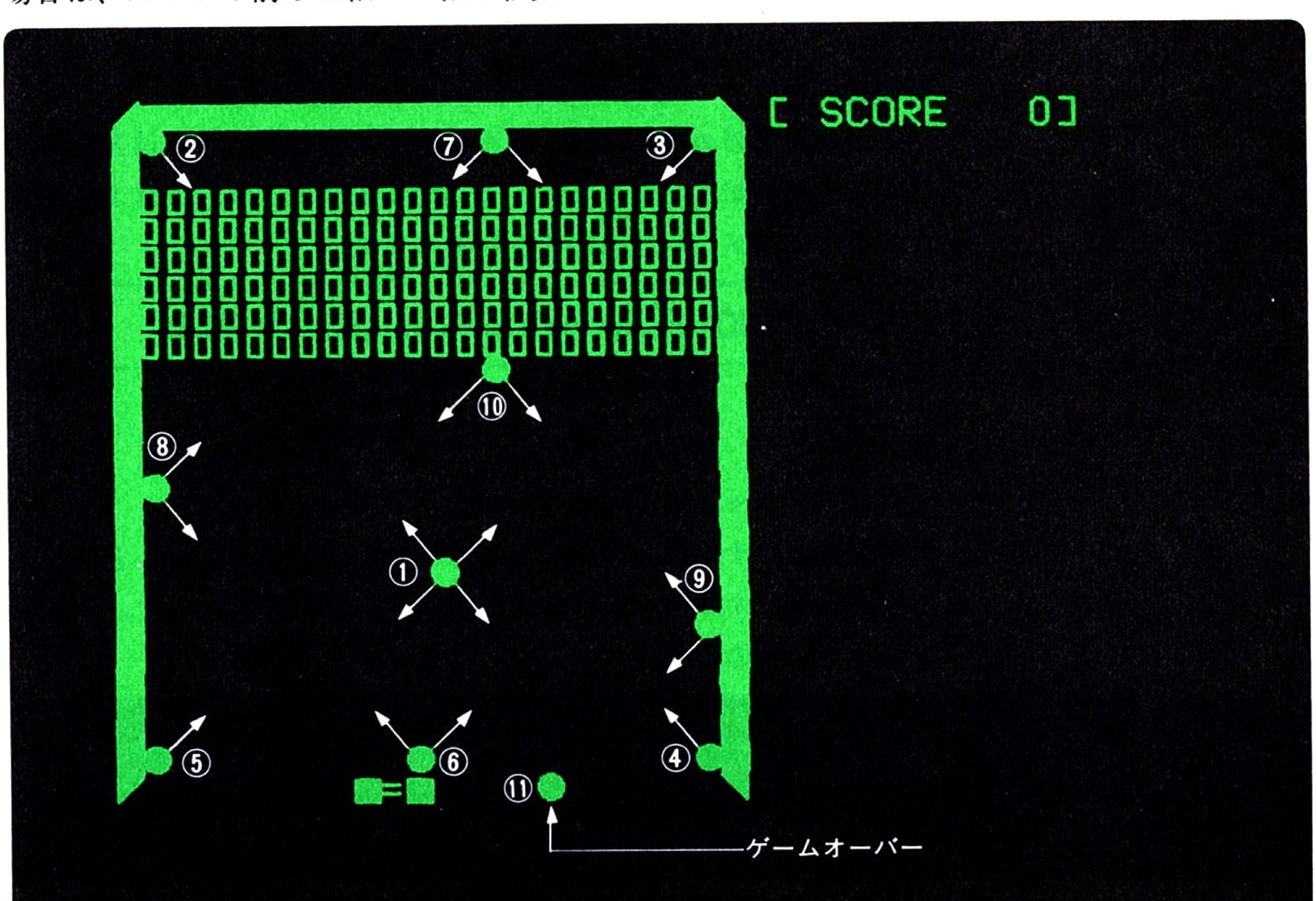


ぶつかる前



ぶつかった後

どうですか？　**X軸方向とY軸方向の速度成分が、ぶつかる前後でまるきり向きが反対**になっているのがわかりますね。ですから、この場合は、ぶつかる前のX軸・Y軸の移動量にマイナスをつけてしまい



ます。

4すみのキャラクタコードは228、229、230、231ですから、これらをまとめて、Pが227よりも大きく(227は含まれません)、Pが232より小さい(232は含まれません)場合に、X軸・Y軸方向の移動量を反転させるようにします。ステートメントは次の通りです。

```
330 IF P>227 AND P<232 THEN MOVE.Y=-MOVE
.Y:MOVE.X=-MOVE.X
```

❻と❼の場合

この場合は、前の例にならって図を描いてみるとわかりますが、X軸方向の成分は変化せず、Y軸方向の成分だけが逆になります。そして、ラケットではねかえってからボールが何かにぶつかるかもしれませんから、すぐにキャラクタチェックのルーチンへ飛ばします。

ラケットの両サイドと天井のキャラクタは同じものを使っていてコードは134、ラケットの中央はイコールを使っていますのでコードは61です。

```
340 IF P=134 OR P=61 THEN MOVE.Y=-MOVE.Y
:GOTO 310
```

❽と❾の場合

ボールが左右のカベにぶつかったときです。この場合にはX軸方向の速度成分だけが逆になります。カベのキャラクタコードは135ですから、ステートメントは次のようになります。

```
350 IF P=135 THEN MOVE.X=-MOVE.X:GOTO 310
```

❿の場合

これは、ブロックをくずしてはねかえる場合です。

BEEP

成功したのですから、ゲームらしく小さな効果音をだすことにしましょう。音をだすコマンドにBEEP(ビープ)があります。ダイレクトモードで次のようにキーインしてみてください。

```
BEEP 1
Ok
█
```

うるさいですね。ピーッという音が鳴り続けています。これを止め

るには、カーソルを「1」のところへ移動して [0] を押してください。

```
BEEP 1 ←────── この数字を0にする
Ok
```

やっと止まりました。このように「BEEP」のうしろに「1」をつけると音が鳴り続け「0」にすれば止まります。また、数字をつけなければ1秒間だけ鳴って止まります。

このBEEP文を使って、いろいろな効果音を作ることができます。次をダイレクトモードでためしてみてください。「日本語ワードプロセッサ」などでモード変換の際によく使うものです。

```
for i=1 to 100:beep 1:beep 0:next
Ok
■
```

みなさんも、いろいろ考えてみてください。

BEEPについてはこのくらいにして、プログラムです。ブロックのキャラクタコードは219ですから、ステートメントを次のようにします。

```
360 IF P=219 THEN BEEP 1:GOSUB *BLOCK
```

「* BLOCK」は、ボールが当たったブロックを消すためのサブルーチンです。

BEEP文は、1というパラメータがついていますから、鳴りっぱなしのはずですが、サブルーチンに飛んでから消音するようにします。

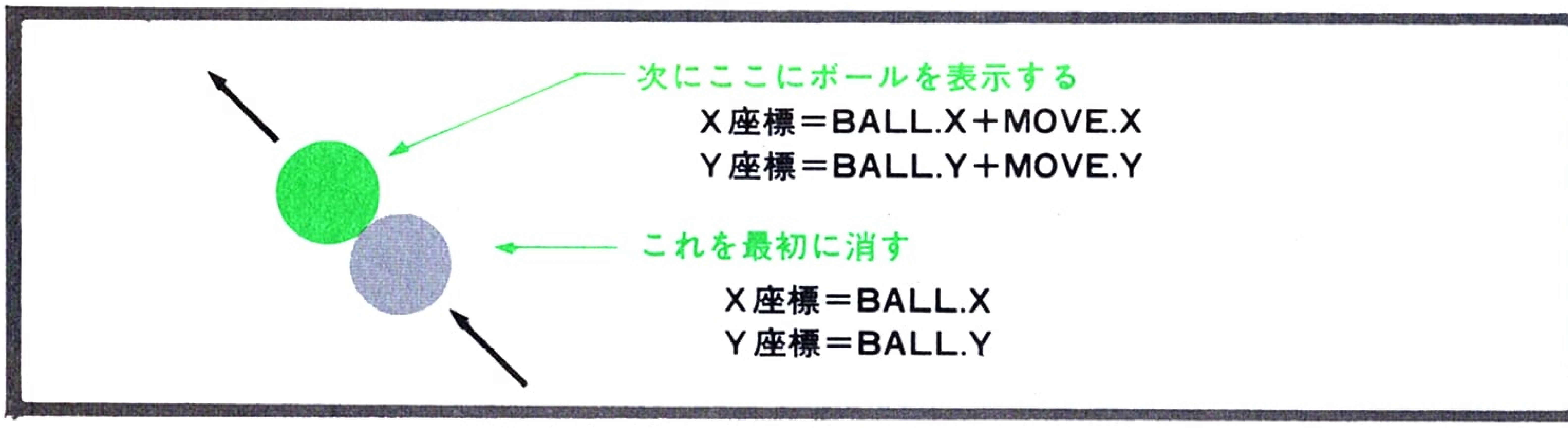
⑪の場合

これはコート外に出てしまった（Y座標が24になってしまった）わけですからゲームオーバー処理に飛ばしますが、ここではステートメントは作りません。というのは、ボールが移動して新しい場所にセットされ、そのY座標が24になっているかどうかを判定してからでないと意味がないからです。

4・5 ボール表示とブロック消去

ボールの表示

行番号 320 で、ボールの動いた位置が空白だったら、そこにボールを表示させなければなりませんね。さっそくボール表示の部分を作ってみましょう。次の図をごらんください。



ボールは新しい座標位置に表示すればよいのですが、上の図からもわかるように、移動前のボールを最初に消しておかなければなりません。そして消去の方法ですが、移動前のボールの位置に空白を出力するようにします。　空白をしまう変数として文字型変数Ｐ＄を使いましょう。

次のステートメントをキーインしてください。

```
400 '
410 *SET.BALL
420 LOCATE BALL.X,BALL.Y:PRINT P$;:BALL.
X=BALL.X+MOVE.X:BALL.Y=BALL.Y+MOVE.Y
```

だいぶ長いステートメントになってしまいましたが、緑色の部分は新しい座標を決めるステートメントですので、行番 425 とでもして、2行にわけてしまってもかまいません。

400 行のＲＥＭ文はリストを見やすくするため、410 行は、行番号320で指定した飛び先のラベルですね。

移動前のボールを消しましたので、新しい位置にボールを表示します。

```
430 LOCATE BALL.X,BALL.Y:PRINT "●";
```

そして変数Pには、キャラクタチェックのサブルーチンでいろいろなキャラクタを調べていますから、どんな数値（コード）が入っているかわかりません。もし、**Pの値がスペースのコード(32)以外でしたら、強制的にPの値を32にしてしまいます。**

32以外で変数Pに入る可能性のある数値は、118ページの表からわかるように61以上の数です。ですから、Pが60よりも大きい場合にPの内容を32にします。

```
440 IF P>60 THEN P=32
```

ここで、ふたたびゲームのメイン部分（行番号230）に戻りたいところですが、ラケットでボールをはねかえしそこねてコートの外へ出てしまう場合も考えられますので、そのチェックをしなければなりません。まだコート内、つまりYが24より小さければ、メイン部分に戻ってゲームを続けられます。

飛ぶ前に、文字型変数P＄の中味も空白にしておきましょう。

```
450 IF BALL.Y<24 THEN P$=CHR$(P):GOTO *M
AIN
```

ゲームオーバー処理

Yの値が24以上（つまり、画面の一番下の行）になってしまったら、ゲームオーバーということにしてメッセージを表示します。

LOCATE文を使って次のようにしてみました。この辺の表示はそれぞれ好みがありますから、みなさん適当に変えてください。

```
460 '
470 LOCATE 7,15:PRINT "*************"
480 LOCATE 7,16:PRINT "  GAME OVER!"
490 LOCATE 7,17:PRINT "*************"
```

Yが24以上ですと、行番号450のIF文の条件が満たされませんから、プログラムの処理は文句なしに行番号460へきてしまいます。

プログラムの実行

さて、ここで4・4節から作ったプログラムリストを示しますので、リストコマンドであなたのリストを画面に表示し、間違いがないか調

べてみてください。

```
310 GOSUB *CHECK
320 IF P=32 THEN GOTO *SET.BALL
330 IF P>227 AND P<232 THEN MOVE.Y=-MOVE
.Y:MOVE.X=-MOVE.X
340 IF P=134 OR P=61 THEN MOVE.Y=-MOVE.Y
:GOTO 310
350 IF P=135 THEN MOVE.X=-MOVE.X:GOTO 310
360 IF P=219 THEN BEEP 1:GOSUB *BLOCK
400 '
410 *SET.BALL
420 LOCATE BALL.X,BALL.Y:PRINT P$;:BALL.
X=BALL.X+MOVE.X:BALL.Y=BALL.Y+MOVE.Y
430 LOCATE BALL.X,BALL.Y:PRINT "●";
440 IF P>60 THEN P=32
450 IF BALL.Y<24 THEN P$=CHR$(P):GOTO *M
AIN
460 '
470 LOCATE 7,15:PRINT "************"
480 LOCATE 7,16:PRINT " GAME OVER!"
490 LOCATE 7,17:PRINT "************"
1000 '
1010 *CHECK
1020 P=PEEK((BALL.Y+MOVE.Y)*120+(BALL.X+
MOVE.X)*2+VRAM)
1030 RETURN
```

それではプログラムをＲＵＮさせて動作テストをしてみましょう。ただし、行番号 360 にアポストロフィを入れてレム文にしておいてください。

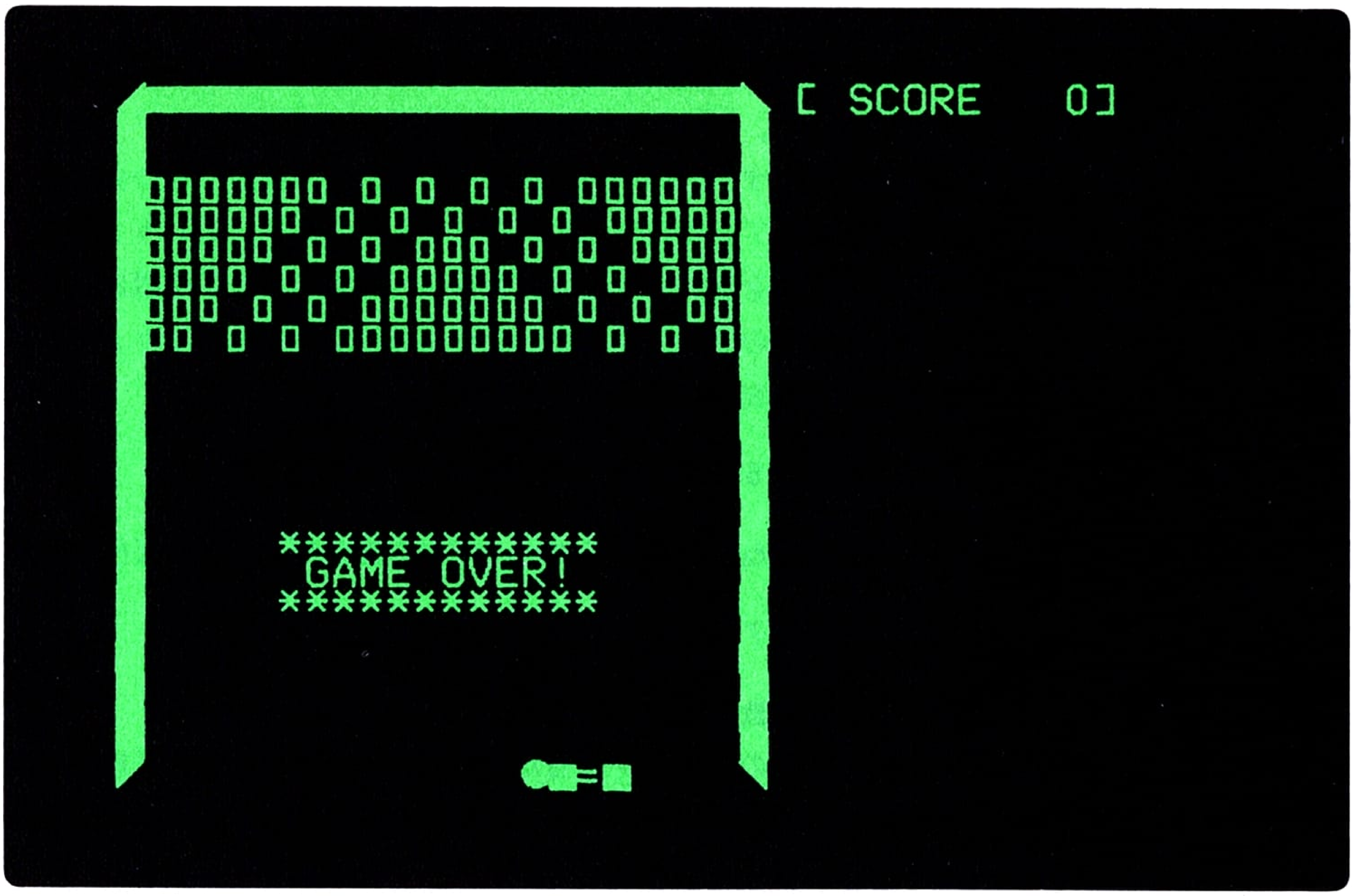
なぜだかわかりますね。ブロック消去のサブルーチンがまだ作ってないからです。

```
360 'IF P=219 THEN BEEP 1:GOSUB *BLOCK
```

↑アポストロフィを入れる

●行番号 500 にＥＮＤ文を入れておいてください。
500　ＥＮＤ

（f･5）キーを押すと、周囲のカベ、ブロックが表示されて、下のほうからボールが飛び出します。はねかえってくるボールをラケットでうまく送り返してください。ブロックにあたっても突き抜けてしまいますが、いまの段階ではこれでよいのです。



1 と 3 のキーを使って、じょうずに操作してください。

上の画面は、ボールのY座標が24になってしまったので、ゲームオーバーになったところです。

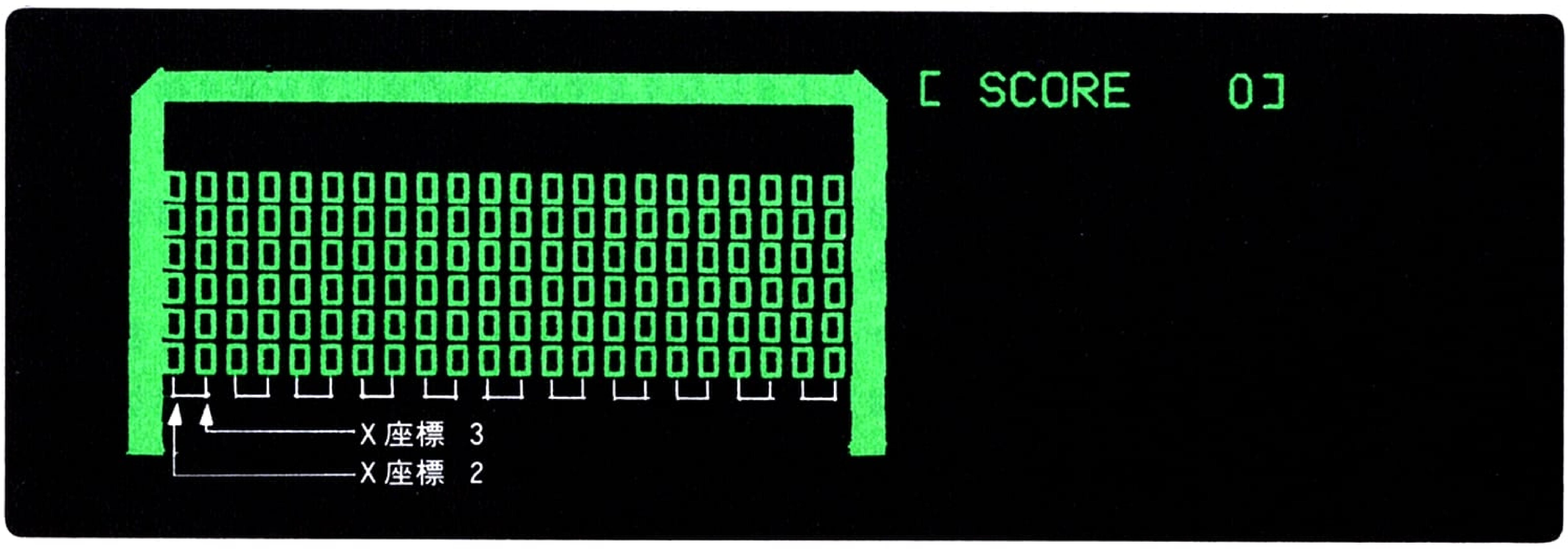
ブロックを消去する

次はブロック消去のサブルーチンです。

行番号360で、もしボールがブロックに当たったら、ブロックを消さなければなりませんね。

ブロックはたくさんありますから、1つずつくずしていくのではちょっと大変です。2ついっぺんに消してしまうことにしましょう。

次の図をごらんください。



図のように2つずつグループを作ってみます。そしてボールが、各グループのどちらに当たっても、同じグループの2つのブロックを消すことにします。

消去の方法ですが、いずれのグループも、その位置に2つ分のスペースをＰＲＩＮＴ文で表示してブロックを消します。ということは、ＬＯＣＡＴＥ文を使いますから、スペースを表示するＸ座標は、どのグループも左側のブロックに合わせておけばよいことになります。

各グループの左側のブロックのＸ座標は

2，4，6，8，10，12，14，16，18，20，22

となっていますね。一方、各グループの右側のブロックのＸ座標は、

3，5，7，9，11，13，15，17，19，21，23

というように奇数になっています。

ですから、どのブロックに当たっても左側のブロックのＸ座標に合わせるということは、ボールのＸ座標

BALL.X＋MOVE.X

を、常に偶数になるようにしておけばよいのです。

この調整をするのに、112ページで使ったＩＮＴ関数を使用します。これは、小数点以下を切り捨てた整数値を与えるものでした。

次のステートメントをダイレクトで入力してみてください。

```
? INT(2/2),INT(3/2)
 1             1
Ok
█
```

これは、一番左のグループを例にとったわけですが、2つのブロックのＸ座標を2で割った値のＩＮＴをとると、いずれも「1」がかえってきます。2を2で割ったときは答は1ですから当然「 1 」になります。3を2で割ると1.5になりますが、小数点以下が切り捨てられて「1」となります。

同じように、2番目、3番目、4番目のグループのＸ座標を2で割ったＩＮＴをとりますと、2、3、4となります。

私たちは、各グループの左側のブロックの座標を求めているのですね。ですから、2で割った値のＩＮＴをとって、それを2倍してしまえば、2、4、6、8というように目的の数値が得られます。

したがって、求めるＸ座標の算式は次のようになります。

```
INT((BALL.X+MOVE.X)/2)*2
```

X座標を2で割る　　2倍する

右向き、左向きのカッコの数を間違えないでください。同数でなければいけませんよ。

さあ、次のステートメントをキーインしましょう。

```
2000 '
2010 *BLOCK
2020 LOCATE INT((BALL.X+MOVE.X)/2)*2,BAL
L.Y+MOVE.Y
```

サブルーチンのラベルは「＊ＢＬＯＣＫ」としました。

ＬＯＣＡＴＥ文によって表示位置が決まりましたので、スペースを出力します。

また、ブロックを１グループくずしたので**スコアを加算**しておきましょう。

```
2030 PRINT "  ";:SCORE=SCORE+1
```

セミコロン、コロンの区別をしっかりしてください。セミコロンをつけ忘れると、このゲームでは画面を25行まるまる使っていますので、上へスクロールしてしまいます。

スコアの加算ができましたから、次は**得点表示**です。

ＰＲＩＮＴ　ＵＳＩＮＧ文を使います。数値を右づめに出力します。

```
2040 LOCATE 33,0
2050 PRINT USING "###0";SCORE
```

変数ＳＣＯＲＥの値がシャープが３つ並んだところに右づめに出力されます。**シャープのうしろの０は、１グループのブロックをくずしたときの得点を10点にするための操作**です。ＳＣＯＲＥに10をかけて表示してもよいのですが、こんな方法も覚えておいてください。

ボールがブロックに当たったということは、121ページの⑩の場合です。進行方向はＹ軸成分が反対になります。ですから、ぶつかる前の移動量にマイナスをつければよいですね。

それから360行でブザー音をだしていましたので、これを消しておきます。

そしてＧＯＳＵＢ文にはつきもののＲＥＴＵＲＮ文です。

次のステートメントをキーインします。

```
2060 MOVE.Y=-MOVE.Y:BEEP 0
2070 RETURN
```

これでブロック消去のサブルーチンは完成です。この部分のリストを次に示します。

```
2000 '
2010 *BLOCK
2020 LOCATE INT((BALL.X+MOVE.X)/2)*2,BAL
L.Y+MOVE.Y
2030 PRINT "  ";:SCORE=SCORE+1
2040 LOCATE 33,0
2050 PRINT USING "###0";SCORE
2060 MOVE.Y=-MOVE.Y:BEEP 0
2070 RETURN
```

もし1面クリアしたら

あなたのキー操作がじょうずで、1面クリアしたらどうしましょうか？　次の2面目へ進みたいですね。ステートメントを追加しましょう。

1面クリアしたかどうかのチェックの方法ですが、行番号360でブロック消去のサブルーチンへ飛んで帰ってきたときに、**変数ＳＣＯＲＥの値がどうであるかによって判断**します。

ブロックは2つ1グループにして全部で66グループありますから、全部くずしたということはＳＣＯＲＥ＝66になっているはずです。ですから、これを条件にしてもよいのですが、知識を吸収する意味で、

ＭＯＤ

ＭＯＤ（モッド）を使ってみましょう。ＭＯＤを使うと、割り算をしたときの余りが求められます。

次の画面をごらんください。

```
? 10 MOD 3.5,10 MOD 3
 2             1
Ok
█
```

割る数が小数の場合には小数点以下を四捨五入した値で演算を行い、その余りを表示します。

そこで、ＳＣＯＲＥがもし66になっていれば、66で割ったときの余りは０になりますね。これを使います。

そして、２回目はブロックが２行分下に表示されて、少しむずかしくなるようにしましょう。ブロックの座標移動を管理する変数に、もう一度、という意味でＡＧＡＩＮを使います。

```
370 IF SCORE MOD 66 =0 THEN AGAIN=AGAIN
+2:GOTO 40 ELSE GOTO 310
```

ＩＦ～ＴＨＥＮ～ＥＬＳＥ

ＩＦ～ＴＨＥＮ～ＥＬＳＥ文がでてきました。これは、ＩＦ文の次の条件が満たされない場合にＥＬＳＥ以下のステートメントを実行します。

上の場合でしたら、「サブルーチンから帰ってきて、もしスコアが66になっていなかったら、行番号40へ飛ぶ」ということになります。

２面目へ進むステートメントを追加しましたので、行番号160の座標を変更しておきましょう。

```
160   LOCATE 2,AGAIN+2+I
```

さあ、それではＲＵＮですが、行番号360をレム文にしておきましたので、アポストロフィをとってから実行してください。

```
360 'IF P=219 THEN BEEP 1:GOSUB *BLOCK
    ↑
    └──────────これをとってください。
```

どうですか？　うまくいきましたか？

●ゲームオーバーになると、画面に１と３が横に表示されるかもしれませんが、いまは気にしないでください。

しかし、何か気づいたことはありませんか？　何度もくりかえしているとわかると思いますが、乱数を発生させるＲＮＤ関数を使っているにもかかわらず、いつも同じような場所からボールが飛びだしてきます。

これはなぜかといいますと、乱数には一定の系列があって、ＲＵＮしたときは、いつも同じところが出発点になってしまうからです。

ですから、ゲームスタート前に乱数をかきまぜて、出発点をずらしておく必要があります。

そこで、ＩＮＫＥＹ＄関数を使い、ゲームスタートの合図があるまで乱数をかきまぜるようにします。

次のようにキーインしてください。

```
35 LOCATE 10,5:PRINT "Hit [S] key to start !"
36 Y$=INKEY$
37 R=RND(1)
38 IF Y$<>"S" THEN GOTO 36
```

●コンピュータには数学で使う不等号（≠）がありませんので「<」「>」を並べて代用します（行番号38）

ＩＮＫＥＹ＄関数は前にでてきましたね。

メッセージにあるように、大文字の「Ｓ」を押さないかぎり、行番号36と38の間をぐるぐる回っています。

そして、その過程で行番37を通りますから、ここで乱数がかきまぜられるというわけです。

メッセージの表示位置は、みなさんが各自調整してください。

アスタリスクでまわりを囲ったり、カラーを変えてみたり、いろいろ工夫できると思います。

また、ゲーム名などを表示してみても楽しいですね。

引き続きゲームをしたいとき

現在、ゲームをするたびに行番号500のＥＮＤ文で終っていますが、自動的にゲームを続けられるようにしましょう。

いままで、プログラムを実行するときに f･5 キーを押していましたが、ＲＵＮコマンドをステートメントにしてしまうのです。

次のようにキーインします。

```
500 '
510 FOR I=1 TO 5000:NEXT
520 RUN
```

行番号510は、みなさんごぞんじのＦＯＲ～ＮＥＸＴ文ですが、これはＦＯＲ文とＮＥＸＴ文の間にステートメントがありませんね。いままでは、数値を表示するとか、何かやるべき仕事がありましたね。

つまり、この場合は、何も仕事をしないで、ＦＯＲ文とＮＥＸＴ文の間を5000回まわるのです。はかってみませんでしたが、５秒くらいかかるのでしょうか。少し道草をくっているわけです。

この形はよく使われるもので、時間つぶしのステートメントです。

次にＲＵＮさせるまでに、少し間をおく、といったところでしょうか。

4・6　プログラムの完成

それでは、前章と同じように、プログラムの前のほうにＲＥＭ文でメモを入れておきましょう。

```
1 ' ﾌﾞﾛｯｸ ｸｽﾞｼ
2 ' file name ==> block
3 ' 1983.11.5 ｻｸｾｲ
4 ' save "block"
5 '
```

カセットにセーブする人は、行番号４は変えなければいけませんね。「ＣＡＳ１：」または「ＣＡＳ２：」をファイル名の前につけるのでした。

次に、見やすくするために、ＲＥＭ文を入れておきましょう。

```
32 '
39 '
```

最後にダイレクトモードで行番号の整理をします。RENUMコマンドを使うのでしたね。

```
renum 100
Ok
renum 1000,710
Ok
renum 2000,1040
Ok
█
```

どのダイレクトステートメントも終りにリターンキーを押さなければいけませんよ。

次ページに「ブロックくずしゲーム」の総プログラムリストをのせておきます。

もし、うまくゲームができないときは、セミコロンをつけ忘れていないか、変数のスペルは正しいか、グラフィックキャラクタに間違いはないか、などをチェックしてください。

```
100 ' ﾌﾞﾛｯｸ ｸｽﾞｼ
110 ' file name ==> block
120 ' 1983.11.5 ｻｸｾｲ
130 ' save "block"
140 '
150 WIDTH 40,25:CONSOLE 0,25,0,1
160 COLOR 6
170 DEFINT A-Z:VRAM=&HF3C8
180 '
190 LOCATE 10,5:PRINT "Hit [S] key to start!"
200 Y$=INKEY$
210 R=RND(1)
220 IF Y$<>"S" THEN GOTO 200
230 '
240 CLS
250 PRINT " ◢";STRING$(22,"■");
260 PRINT "◣ [ SCORE    0]"
270 FOR I=1 TO 23
280   LOCATE 1,I:PRINT "■"
290 NEXT I
300 FOR I=1 TO 23
310   LOCATE 24,I:PRINT "■"
320 NEXT I
330 LOCATE 1,24:PRINT "◤";SPC(22);"◥";
340 COLOR 3
350 FOR I=1 TO 6
360   LOCATE 2,AGAIN+2+I
370   PRINT STRING$(22,"□")
380 NEXT I
390 BALL.Y=23:MOVE.Y=-1:LIMIT=10
400 BALL.X=RND(1)*22+2
410 MOVE.X=INT(RND(1)*2)*2-1
420 '
430 *MAIN
440 I=INP(0)
450 COLOR 5:BEEP 0
460 RACKET.X=(I=247)*(LIMIT<21)-(I=253)*
(LIMIT>2)
470 LOCATE LIMIT,24:PRINT "   ";
480 LIMIT=LIMIT+RACKET.X
490 LOCATE LIMIT,24:PRINT "■=■";
500 COLOR 7
```

- 150–170: 初期設定
- 200–220: 乱数のかきまぜとキー入力待ち
- 240: 画面クリア
- 250–330: カベをつくる
- 340–380: 紫色でブロックを表示
- 390–410: ボールとラケットの初期座標
- 440: キー入力を知るための関数
- 460–490: ラケットを表示
- 500: 白色にする

```
510 GOSUB *CHECK
520 IF P=32 THEN GOTO *SET.BALL
530 IF P>227 AND P<232 THEN MOVE.Y=-MOVE
.Y:MOVE.X=-MOVE.X
540 IF P=134 OR P=61 THEN MOVE.Y=-MOVE.Y
:GOTO 510
550 IF P=135 THEN MOVE.X=-MOVE.X:GOTO 510
560 IF P=219 THEN BEEP 1:GOSUB *BLOCK
570 IF SCORE MOD 66 =0 THEN AGAIN=AGAIN+
2:GOTO 240 ELSE GOTO 510
580 '
590 *SET.BALL
600 LOCATE BALL.X,BALL.Y:PRINT P$;:BALL.
X=BALL.X+MOVE.X:BALL.Y=BALL.Y+MOVE.Y
610 LOCATE BALL.X,BALL.Y:PRINT "●";
620 IF P>60 THEN P=32
630 IF BALL.Y<24 THEN P$=CHR$(P):GOTO *M
AIN
640 '
650 LOCATE 7,15:PRINT "************"
660 LOCATE 7,16:PRINT " GAME OVER!"
670 LOCATE 7,17:PRINT "************"
680 '
690 FOR I=1 TO 5000:NEXT
700 RUN
1000 '
1010 *CHECK
1020 P=PEEK((BALL.Y+MOVE.Y)*120+(BALL.X+
MOVE.X)*2+VRAM)
1030 RETURN
2000 '
2010 *BLOCK
2020 LOCATE INT((BALL.X+MOVE.X)/2)*2,BAL
L.Y+MOVE.Y
2030 PRINT "  ";:SCORE=SCORE+1
2040 LOCATE 33,0
2050 PRINT USING "###0";SCORE
2060 MOVE.Y=-MOVE.Y:BEEP 0
2070 RETURN
```

キャラクタコードのチェック（510）

キャラクタコードにより各飛び先へ（520～570）

ボールを表示（600～630）

ちょっと一息（690）

プログラムの実行（700）

ボールの動いたところに何があるか（1020）

ブロックを消して得点加算（2020～2030）

スコア表示（2040～2050）

第5章

算数教室

5・1　プログラムのあらすじ

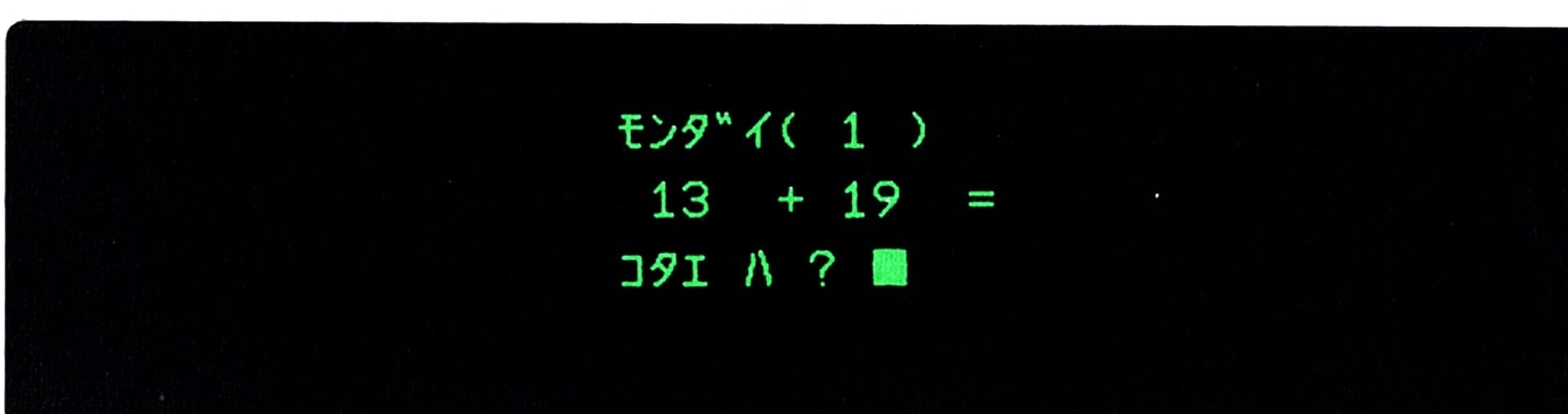
「算数教室？　さえないなあ」なんていわないでくださいよ。
前章でゲームを楽しんだのですから、ここではちょっとマジメにプログラミングをしましょう。

このプログラムは小学低学年用のもので、たし算、ひき算、九九を扱います。
ですから、お子さんがいらっしゃる方は、学習用に役立ててください。また、そうでない方も、プログラミングテクニックの勉強と思ってがんばってください。

プログラムの流れですが、まず次のような見出しを表示します。



次に、問題メニューを表示して、「たし算」「ひき算」「九九」のいずれかを選択させます。
選択番号によって各サブルーチンへ飛び、次のような形で問題をだします。



このようにして10問出題され、最後に得点を表示するようにします。

これがプログラムのあらすじです。
あるテーマについてのプログラムを作るとき、このように、おおざっぱな流れを頭に描き、製作にとりかかるとよいでしょう。

5・2　初期設定

初期設定

画面の桁数および行数は40桁×25行、カラーモードにして、ファンクションキーの表示はしないことにします。

この部分は前章とまったく同じです。

```
100 CONSOLE 0,25,0,1:WIDTH 40,25
```

次に、すべての変数の整数宣言をしておきましょう。

```
110 DEFINT A-Z
```

プログラムの先頭では、まず画面をクリアします。また、見出しは黄色で表示することにしましょう。次のステートメントをキーインします。

```
120 CLS:COLOR 6
```

これで初期設定は終りです。けっこう簡単ですね。

見出しの表示

次は見出しの表示です。

前ページの画面のように「サンスウ　キョウシツ」というタイトルを飾りケイで囲います。

このタイトルをＰＲＩＮＴ文で表示しなければなりませんが、復習の意味もかねて第３章の方法でやってみましょう。

そして、この部分はサブルーチンにしますので、次のステートメントを入力します。

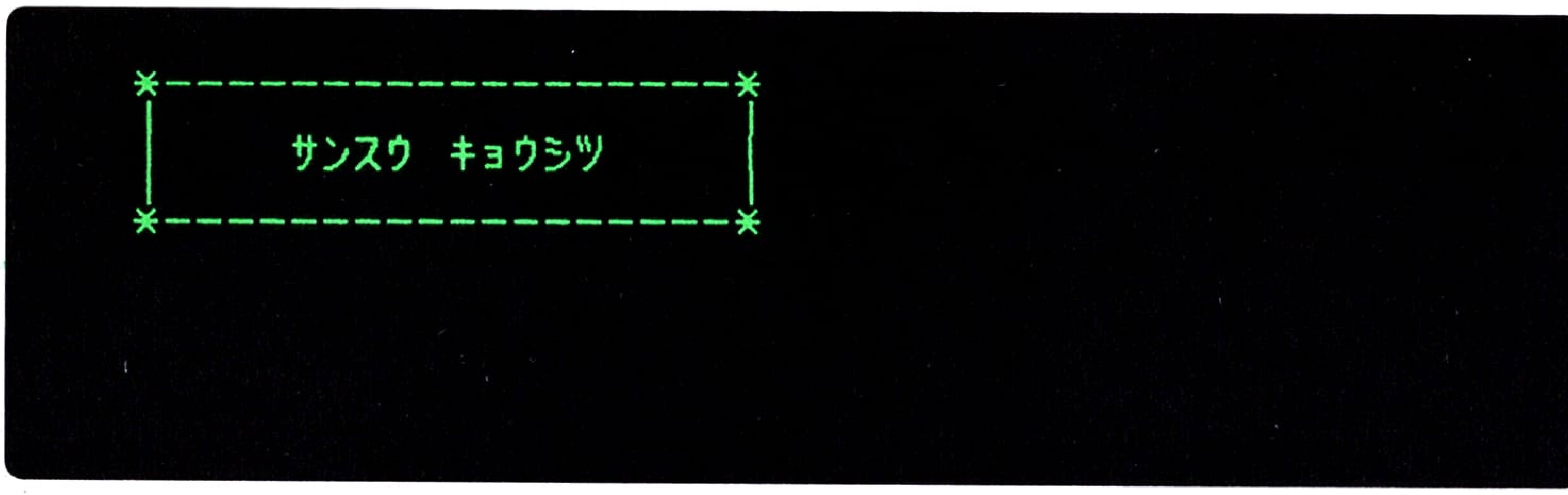
```
130 GOSUB *MIDASHI
```

そうしましたら、サブルーチンへ飛ぶときコンピュータが識別するラベルが必要ですね。

```
1000 '
1010 *MIDASHI
```

さあ、これでサブルーチンの下準備ができました。前ページの画面と同じタイトルをグラフィックキャラクタ、カタカナを使って画面に

描いてください。ホームクリアキーで画面をクリアして、ホームポジションの位置から描きはじめましょうか。
画面モードは、どんなモードでもかまいません。

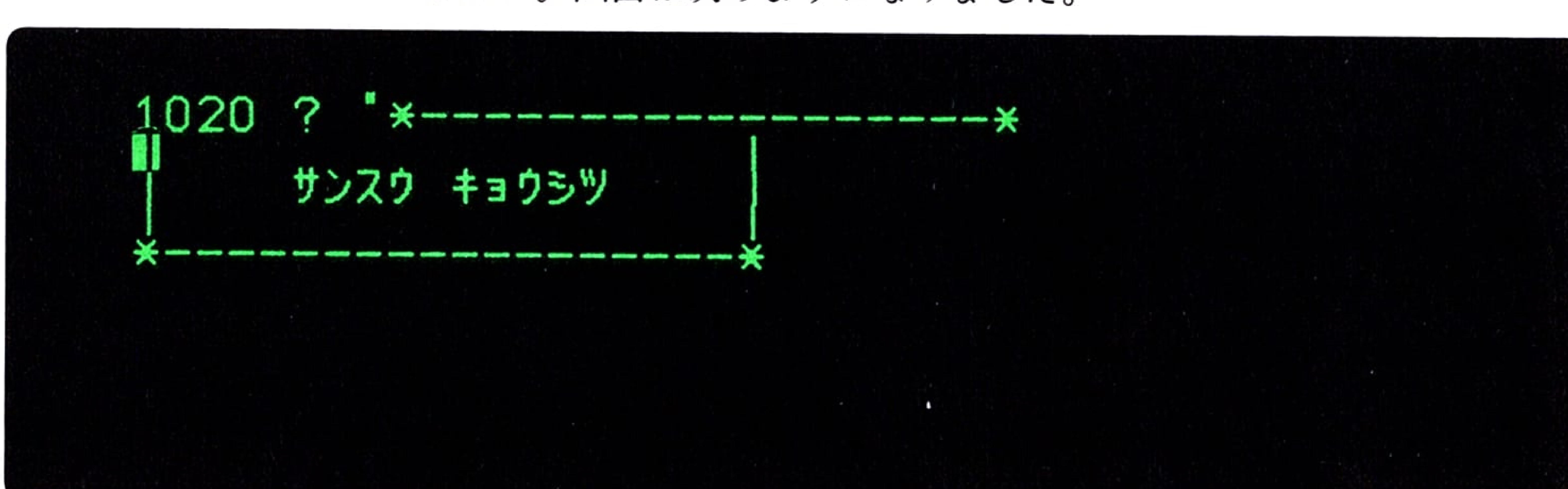


上下のケイ線はマイナス符号を、左右のケイ線は GRAPH キーとテンキー側の＝キーを同時に押したときのキャラクタです。
レイアウトは、みなさんのほうがじょうずだと思います。もっと見ばえのする飾りつけを考えてください。

図が描き上がりましたら、カーソルを移動して左すみのアスタリスクに重ねてください。この操作は第３章の復習ですよ。すでにマスターできている方は、自分で進めていってください。
カーソルがアスタリスクに重なりましたら、 SHIFT キーを押しながら INS キーを一度押します。これでインサートモードになりましたね。
そうしたら、次の順でキーをたたいてください。

1　0　2　0　スペース　?　スペース　"　↵

クォーテーションマークをカタカナの「だく点」と間違わないでください。画面は次のようになりました。



同じ操作をくりかえして一番下の行までＰＲＩＮＴ文を作ってください。スペースとかクエッションマークを入れずにリターンキーを押してしまったら、もう一度、その行にカーソルを戻して、再びイン

サートモードにして挿入するようにしてください。要は、間違えてもあまりあせらないことです。

スクリーンエディトが終ったら、画面をクリアしてＬＩＳＴコマンドで確認しておきましょう。

```
list 1020-1060
1020  PRINT "*---------------------*
1030  PRINT "|                     |
1040  PRINT "|     ｻﾝｽｳ  ｷｮｳｼﾂ     |
1050  PRINT "|                     |
1060  PRINT "*---------------------*
Ok
█
```

上のＰＲＩＮＴ文では画面の左すみから表示されてしまいますから、ＬＯＣＡＴＥ文を使ってちょうどよい位置に移動しましょう。

●上の画面で、行番号とＰＲＩＮＴの間に２スペース入っていますが、１字分のスペースでもかまいません。これはプログラムリストの体裁上の問題ですから。

ＰＲＩＮＴ文のすぐ前にステートメントをつければよいわけですから、行番号は1015から始めて10きざみにします。

これらのステートメントを入力する前に、新しい知識を１つマスターしましょう。それは、ファンクションキーをコンピュータの定義に従ってではなく、私たちが定義して自由に使うことです。

あなたのディスプレイに、現在ファンクションキー表示はでていますか？　もし表示されていませんでしたら、

```
console ,,1 ⏎
```

とキーインしてみてください。ファンクションキーが表示されましたね。ＣＯＮＳＯＬＥ文の３番目のパラメータを１にすればよいのです。前の２つのパラメータを省略した場合は、スクロール開始行とスクロール行数は現在のままです。ただし、コンマを省略してはいけませんよ。

ファンクションキー表示の左から右へ f･1 キーから f･5 キーが順番に対応しています。

この中で、行番号を自動的に発生させる「auto」というのはあまり使いませんので、これに対応した f･2 キーに私たちがよく使うコマンドを定義しましょう。

ユーザーによるファンクションキー定義

次のようにダイレクトモードでキーインしてください。

```
key 2,"LOCATE "
```

（"LOCATE" の後に）スペースを１つ入れる

●ＰＣ8001ではコマンドとパラメータの間にスペースを入れずに打ち込んでもエラーにならなかったのですがＰＣ8801ではうまくいきません。これはスペースを区切りとして使っているためです。

「ｋｅｙ」と「２」の間にスペースを１つ入れませんと Syntax error になりますよ。このエラーは構文上のエラーで、書式に合っていないということです。一番多く現れるエラーです。みなさんもこれまでに何回かお目にかかったのではないですか？　このエラーがでたときは、まず第１に、本のリストと画面上のリストを１字１字くらべるようにしてください。

```
key2,"LOCATE "
Syntax error
Ok
```

さて、ファンクションキー表示の左から２番目を見てください。いままで「auto」だったのが「ＬＯＣＡＴＥ」に変わっていますね。

では f･2 キーを一度押してみてください。「LOCATE」と画面に表示されるでしょう。

●ユーザーの指定したファンクションキー定義は、電源を切れば消えてしまいます。また、最大15文字までの文字列およびコントロール文字を定義できます。

このように、ＬＯＣＡＴＥ文をよく使う場合には、ファンクションキーに定義しておけば、１字１字キーインする必要はなく、そのキーを押すだけでよいのです。

ファンクションキーの定義にはおもしろい使い方もありますが、ここでは説明をはぶきます。

f･2 キーに「LOCATE　」を定義しましたので、これを使って、見出し表示位置のステートメントをキーインしてみましょう。

たとえば行番1015でしたら、

1 0 1 5 スペース f･2 9 , 8 ↵

です。

同じように、次の各行を入力してください。

```
1025  LOCATE 9,9
1035  LOCATE 9,10
1045  LOCATE 9,11
1055  LOCATE 9,12
```

これで、見出し表示のサブルーチンは完成したわけですから、最後にＲＥＴＵＲＮ文でしめくくりたいところですが、何秒かこのままの状態で表示させておき、その後画面をクリアして飛び先へ戻るように

しましょう。

前章でマスターした時間待ちのステートメントを使います。

```
1070  FOR I=1 TO 5000:NEXT I
1080  CLS
```

待ち時間の長短はみなさんが各自調整してください。

そして最後にRETURN文です。

```
1090 RETURN
```

それでは f･5 キーでRUNさせてみてください。タイトルは表示されましたが、次のようなエラーメッセージが現れました。

```
RETURN without GOSUB in 1090
Ok
■
```

これは「GOSUB文がないのにRETURN文がありますよ」というあなたへの注意です。1000行からのサブルーチンが2回実行され、2回目の実行がGOSUB文で飛んでいったためではなかったので生じたエラーです。

●一度GOSUB文を実行しますと、処理は1000行にに移ります。

では、この節のプログラムリストです。

```
100 CONSOLE 0,25,0,1:WIDTH 40,25
110 DEFINT A-Z
120 CLS:COLOR 6
130 GOSUB *MIDASHI
1000 '
1010 *MIDASHI
1025  LOCATE 9,8
1020  PRINT "*--------------------*
1025  LOCATE 9,9
1030  PRINT "|                    |
1035  LOCATE 9,10
1040  PRINT "|     ｻﾝｽｳ  ｷｮｳｼﾂ     |
1045  LOCATE 9,11
1050  PRINT "|                    |
1055  LOCATE 9,12
1060  PRINT "*--------------------*
1070  FOR I=1 TO 5000:NEXT I
1080  CLS
1090 RETURN
```

5・3 メニュー画面

プログラム作成中の画面モード

このプログラムを一度ＲＵＮさせますと、40桁×25行のカラーモードになってしまいます。カラー文字は少々疲れると思われる方は、次のステートメントをダイレクトで入力してください。

```
color 7
```

← 文字を白に

80桁モードのほうが見やすいという方は、

```
width 80
```

と入力してください。

以後、**プログラムを実行して再びプログラムの製作に移るたびに、みなさんのお好きなモードに切り換えてください。**

メニューを表示する

タイトルの表示が終りましたら、いよいよ算数のレッスンです。

まず、次のような**メニュー**を表示することにします。

```
(9，7)
ト゛ノ モンタ゛イ ヲ ヤリマスカ ？

   1・・・タシサ゛ン
   2・・・ヒキサ゛ン
   3・・・クク

  ハ゛ンコ゛ウ ヲ イレテ RETキー ？ ■
```

画面に示しましたように、「ド」のところが座標(9，7)になっています。ＬＯＣＡＴＥ文とＰＲＩＮＴ文を組み合わせてステートメントを作ってみましょう。

メニュー番号の後ろの点は、ピリオドではなく、(カナ)キーをロックしてから(SHIFT)キーと(メ)のキーを押したとき表示されるものです。

●ＬＯＣＡＴＥとキーインするときは先ほど定義したf･2キーを使いましょう。

```
140 LOCATE 9,7 :PRINT "ト゛ノ モンタ゛イ ヲ ヤリマスカ ?"
150 LOCATE 9,9 :PRINT "  1･･･タシサ゛ン"
160 LOCATE 9,10:PRINT "  2･･･ヒキサ゛ン"
170 LOCATE 9,11:PRINT "  3･･･クク"
```

次に番号選択のメッセージを入れますが、これは入力を要するわけですから、ＰＲＩＮＴ文ではなく**ＩＮＰＵＴ文**を使います。

そして、このメッセージは、注意をうながす意味で**赤色で表示**することにしましょう。赤のカラーコードは２でしたね。

```
180 COLOR 2
190 LOCATE 9,13:INPUT " ハ゛ンコ゛ウ ヲ イレテ RET
キー";NUMBER
```

行番号190にメッセージのうしろの「？」は入れる必要はありませんよ。**ＩＮＰＵＴ文は実行すれば必ずクエッションマークが表示される**ようになっています。

これでメニュー選択ができました。間違って違う数値を入力することも考えられますので、**制限**をしておきましょう。

```
200 IF NUMBER>3 THEN BEEP:COLOR 6:GOTO 1
40
```

（40 → 行番号ではありません。前の行から続けてキーインしてください）

入力値が３より大きければ、ブザー音が鳴って、振り出しに戻ることになります。

(f･5)キーで実行してみましょう。

タイトルが表示されてメニュー画面がでます。そして、番号入力をうながしています。ここで、４以上の数を入力してリターンキーを押してみてください。再びメニュー画面に戻りましたね。

うまく動作しているようです。(STOP)キーでプログラムの実行を止めて、次に進みましょう。

5・4　各サブルーチンへ分岐

いま入力されたメニュー番号によって、「たし算」「ひき算」「九九」のサブルーチンへ飛ばしたいのですが、次のように条件文を３つ並べても間違いではありませんが、ここではもっと便利な方法を使います。

次のステートメントは入力しないでください。

```
210 IF NUMBER=1 THEN GOSUB *TASHIZAN
220 IF NUMBER=2 THEN GOSUB *HIKIZAN
230 IF NUMBER=3 THEN GOSUB *KUKU
```

これらの３つの行を１つにするステートメントがあります。それはＯＮ～ＧＯＳＵＢ文です。書式は次の通りです。

```
ON A GOSUB 1000,2000,3000
```

ＯＮ～ＧＯＳＵＢ

Ａには１から始まる整数が入り、ＧＯＳＵＢ以下の行番号の並びがこれに対応しています。

この書式でいいますと、Ａ＝２だったら行番2000からの、Ａ＝３だったら行番号3000からのサブルーチンにジャンプします。

行番号はラベルにおきかえてもかまいませんので、上の３つのステートメントは次の220行のように１つにまとめることができます。

```
210 CLS
220 ON NUMBER GOSUB *TASHIZAN,*HIKIZAN,*
KUKU
```

ラベル名はわかりやすいように具体的にしました。

１、２、３の入力に対して、それぞれ「たし算」「ひき算」「九九」のサブルーチンに分岐することになります。

それでは、上のようにキーインしてください。

ＯＮ～ＧＯＴＯ

同じ使い方に**ＯＮ～ＧＯＴＯ文**がありますので、これも頭に入れておきましょう。書式はまったく同じです。

```
ON A GOTO 1000,2000,3000
```

意味もＯＮ～ＧＯＳＵＢ文と同じで、異なるのはもとに戻ってこないで行きっぱなしという点です。

5·5 たし算のサブルーチン

メニュー画面で１を選んだ場合、このルーチンに飛んできます。
レイアウトは、おおよそ下の画面のようにしたいと思います。
まず、サブルーチンの入口は、次のようになります。

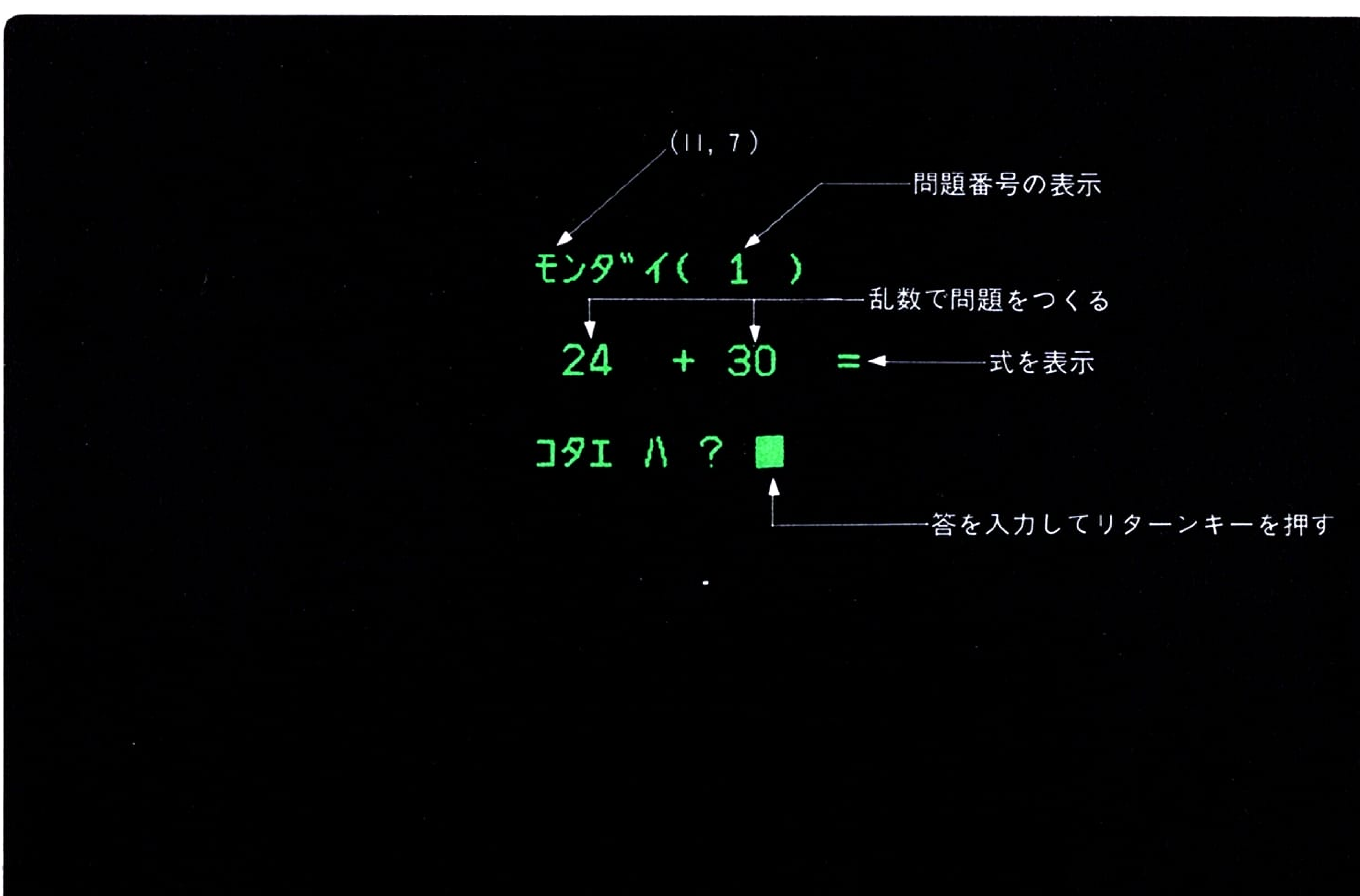
```
2000 '
2010 *TASHIZAN
2020  COLOR 4
```

この部分のカラーは緑にしましょう。子ども相手だとカラフルなほうがよいでしょう。

出題数は毎回10回と決めておきましょう。同じことを10回くりかえしますので、ＦＯＲ～ＮＥＸＴ文を使います。

ＦＯＲ～ＮＥＸＴループの中で、まず最初にすることは、問題番号の表示です。ＬＯＣＡＴＥ文とＰＲＩＮＴ文の組み合わせで作れますね。

```
2030  FOR I=1 TO 10
2040   LOCATE 11,7:PRINT "ﾓﾝﾀﾞｲ(";I;")"
```



問題は１番から順番に出題されますから、ループ変数Ｉを使います。

ＰＲＩＮＴ文で文字列を出力するときは、ダブルクォーテーションでくくります。数値変数の場合はそのままです。次のようにしてしまうと、番号は表示されませんから注意してください。

```
2040 PRINT "ﾓﾝﾀﾞｲ(I)"
```

変数をダブルクォーテーションの中に入れてしまった

この場合には、数値変数のはずのＩが文字と考えられてしまい、Ｉというアルファベットが表示されてしまいます。

次は出題部分です。このプログラムは、２桁のたし算に限定することにしますが、２数をどのように作りだしたらよいでしょうか。

いつも同じ数ばかりではしかたありませんから、ここは、前に勉強した乱数関数ＲＮＤを使うことにしましょう。そして、この２数を作りだすルーチンは、ひき算、九九でも必要になりますので、サブルーチン化しておきます。ラベル名を「＊ＫＡＺＵ」として次のようにキーインします。

```
2050    GOSUB *KAZU
```

２数を発生させるサブルーチンは、あとで作ることにして、ＦＯＲ～ＮＥＸＴループを完成させてしまいましょう。

サブルーチンで作る２数のための変数を仮りに

ＫＡＺＵ１，ＫＡＺＵ２

としておきますと、答えは、この２つの変数の中味を加算したものになりますから、その結果を「答え」用の変数

ＫＯＴＡＥ

に代入します。次のステートメントを追加しましょう。

```
2060    KOTAE=KAZU1+KAZU2
```

変数ＫＯＴＡＥの中味が２数の加算結果ですから、これでコンピュータは答えを記憶することができました。コンピュータを先生にたとえれば、次は回答者、つまり生徒が計算する番です。

しかし、まだ画面には問題がだされていません。これを表示しないことには、どんな秀才だって答えようがありませんね。

ＬＯＣＡＴＥ文で位置を決めて、ＰＲＩＮＴ文で問題を表示します。回答をうながすメッセージもＩＮＰＵＴ文を使って入れておきます。

```
2070    LOCATE 11,9
2080    PRINT KAZU1;"+";KAZU2;" ="
2090    LOCATE 11,11:INPUT "ｺﾀｴ ﾊ ";ANSER
```

プラス記号とイコールは文字列ですから、ダブルクォーテーションマークでくくってください。

さて、これで先生であるコンピュータからの問題がそろいました。そして画面には「コタエ　ハ？」と表示されるはずですから、生徒はここで答を入力しなければなりません。

答をキーインし、間違いないと思ったらリターンキーを押します。

リターンキーが押されると、コンピュータはそれをキャッチし、すぐに答が正しいかどうかチェックしなければなりません。

生徒の答は変数ＡＮＳＥＲに、先生の答は変数ＫＯＴＡＥに入っています。ですから、この２つの変数の中味が等しいかどうか比較すればよいわけです。

そして、等しければ当たった場合の処理を、等しくなければはずれた場合の処理をします。

当たった場合には画面に「アタリ」と、はずれた場合には「**ハズレ**」と大きく表示するようにしましょう。子どもも相手だといろいろ工夫して、だましだましやらせないと、すぐあきてしまいますからね。

この「アタリ」と「ハズレ」の処理はサブルーチンにやらせます。

次のステートメントを入力してください。

```
2100     IF ANSER=KOTAE THEN GOSUB *ATARI
:SEIKAI=SEIKAI+1:GOTO 2120
2110    GOSUB *HAZURE
2120    CMD CLS
```

当たったときは、変数ＳＥＩＫＡＩの中味を１ずつ加算し、得点表示のときの材料にします。

はずれの場合は、2100行にはひっかかりませんから、2110行でサブルーチンに飛ばします。

いずれの場合も、サブルーチンでメッセージ表示がありますので、帰ってきたら画面クリアをしておきます。

ここまでで、ＦＯＲ～ＮＥＸＴの１回のループが終了します。再びＦＯＲ文に戻らせるために、次のＮＥＸＴ文を入れます。

```
2130   NEXT I
```

同じことを10回くりかえしたら、ＦＯＲ～ＮＥＸＴループを抜け出

し飛び先へ帰らなければなりませんので、次のＲＥＴＵＲＮ文が必要ですね。

```
2140 RETURN
```

これでたし算のサブルーチンが完成しました。リストを示します。

```
2000 '
2010 *TASHIZAN
2020  COLOR 4
2030  FOR I=1 TO 10
2040   LOCATE 11,7:PRINT "ﾓﾝﾀﾞｲ(";I;")"
2050   GOSUB *KAZU
2060   KOTAE=KAZU1+KAZU2
2070   LOCATE 11,9
2080   PRINT KAZU1;"+";KAZU2;" ="
2090   LOCATE 11,11:INPUT "ｺﾀｴ ﾊ ";ANSER
2100    IF ANSER=KOTAE THEN GOSUB *ATARI
        :SEIKAI=SEIKAI+1:GOTO 2120
2110   GOSUB *HAZURE
2120   CLS
2130  NEXT I
2140 RETURN
```

たす数とたされる数を作る

2050行で、２数を作りだすサブルーチンへ飛ばすステートメントが入れてありますので、これを用意しておきましょう。

ＲＮＤ関数を使いますが、２桁の数という制限がありますので、発生する乱数に100をかけて、その整数部分をとっておきます。ＩＮＴ関数を使えばよいですね。

行番号を5000行からとして、次のように入力しましょう。

```
5000 '
5010 *KAZU
5020  KAZU1=INT(RND(1)*100)
5030  KAZU2=INT(RND(1)*100)
5040 RETURN
```

5・6 「アタリ」「ハズレ」の画面表示

「アタリ」と表示する

キー入力した答が正しかった場合には、このサブルーチンに飛んできて、ブザー音を鳴らして、下の画面のようなメッセージを表示するようにしましょう。

これも、5・2節と同じようにカーソルを動かしながらグラフィックキャラクタを使って画面に文字を描き、それをＰＲＩＮＴ文にします。

みなさんはもう方法はわかりますね。やってみてください。

ただその前に、サブルーチンのラベル、カラー、ブザー音などのプログラムを作っておきましょう。

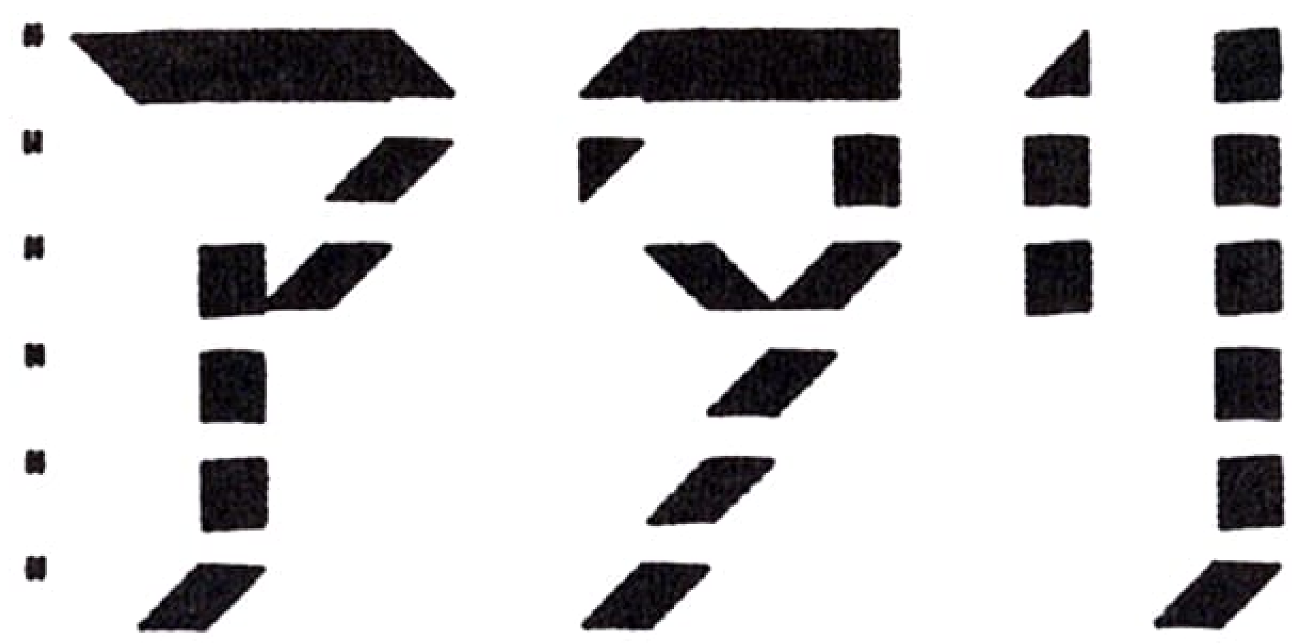
```
6000 '
6010 *ATARI
6020  CLS:COLOR 1
6030  FOR I=1 TO 200:BEEP 1:BEEP 0:NEXT I
```

一度画面をクリアしてからカラー指定をし、ブザー音を入れます。あまり、「アタリ」らしくない音かもしれませんので、あなたなりに調整してみてください。文字の色は青色にしました。



前ページの画面から作った、「アタリ」表示の部分は次のようになります。

```
6040  PRINT "
6050  PRINT "
6060  PRINT "
6070  PRINT "
6080  PRINT "
6090  PRINT "
```



そして、これにＬＯＣＡＴＥ文をつけ加えます。

```
6035  LOCATE 9,5
6045  LOCATE 9,6
6055  LOCATE 9,7
6065  LOCATE 9,8
6075  LOCATE 9,9
6085  LOCATE 9,10
```

このメッセージは、短い時間まち（タイマー）を入れ、２秒くらい画面に表示しておきます。次にＲＥＴＵＲＮ文で飛び先に帰る前にカラーを緑に戻しておきます。「ハズレ」の場合は、同様にして各自作ってみてください。リストを右ページに示します。



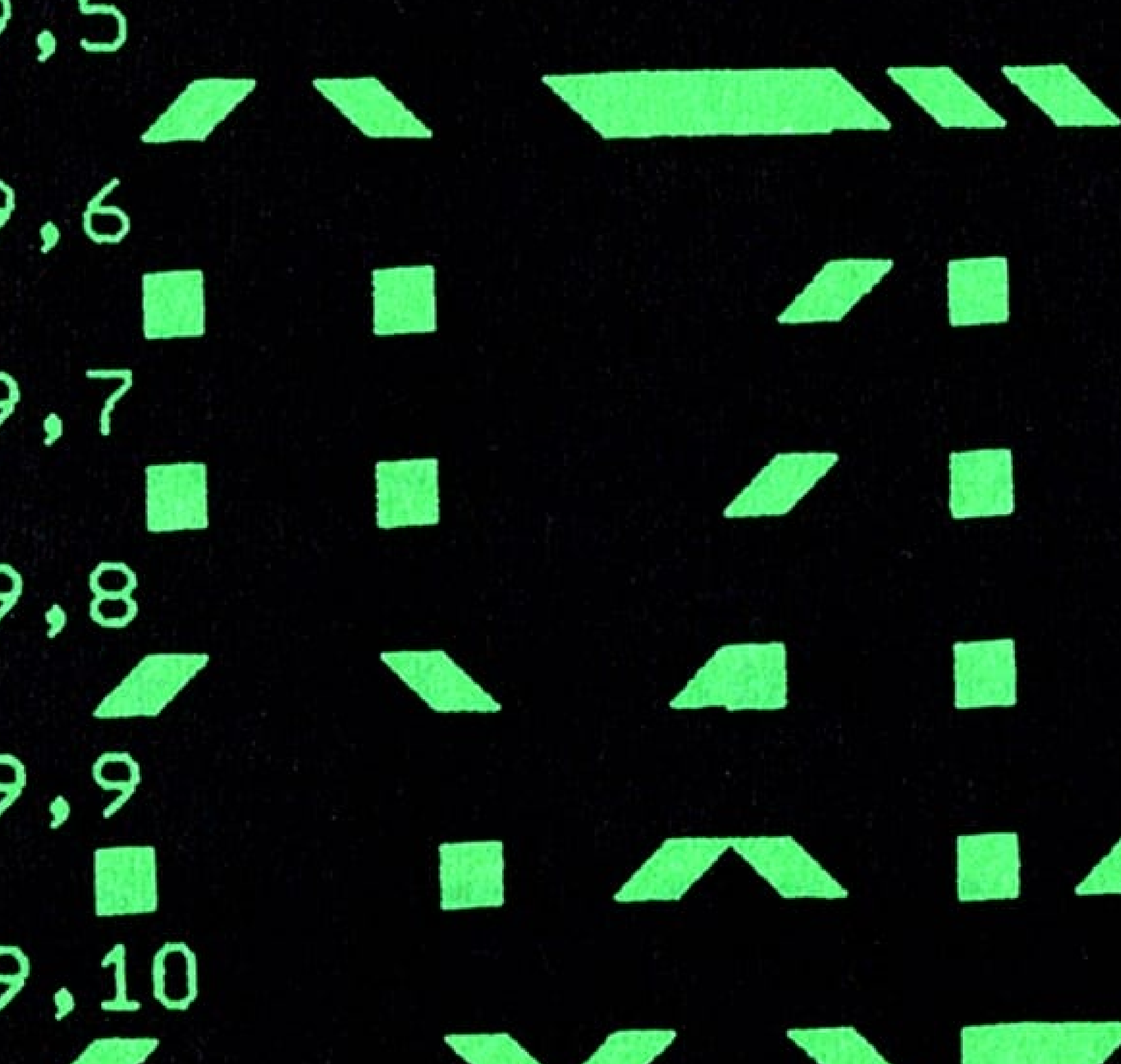
```
6100  FOR I=1 TO 1000:NEXT I
6110  COLOR 4
6120 RETURN
```

次は、「アタリ」「ハズレ」の総リストです。

```
6000 '
6010 *ATARI
6020  CLS:COLOR 1
6030  FOR I=1 TO 200:BEEP 1:BEEP 0:NEXT I
6035  LOCATE 9,5
6040  PRINT "
6045  LOCATE 9,6
6050  PRINT "
6055  LOCATE 9,7
6060  PRINT "
6065  LOCATE 9,8
6070  PRINT "
6075  LOCATE 9,9
6080  PRINT "
6085  LOCATE 9,10
6090  PRINT "
6100  FOR I=1 TO 1000:NEXT I
6110  COLOR 4
6120 RETURN
```



```
7000 '
7010 *HAZURE
7020  CLS:COLOR 2
7030  BEEP:BEEP:BEEP
7035  LOCATE 9,5
7040  PRINT "
7045  LOCATE 9,6
7050  PRINT "
7055  LOCATE 9,7
7060  PRINT "
7065  LOCATE 9,8
7070  PRINT "
7075  LOCATE 9,9
7080  PRINT "
7085  LOCATE 9,10
7090  PRINT "
7100  FOR I=1 TO 1000:NEXT I
7110  COLOR 4
7120 RETURN
```



5・7 ここで動作テスト

さあ、だいたいの骨組ができあがりました。ここまでのプログラムを一度実行してテストをしてみましょう。

メニュー番号は1を選択してください。

(f・5) キーを押すと次の順でプログラムが進行します。

❶ 「サンスウ　キョウシツ」のタイトルがでる。

❷ 問題メニューが表示される。1をキーインしてリターンキーを押す。

❸ モンダイ(1)の出題があるので答を入力してリターンキーを押す。

❹ 正解をキーインすれば「アタリ」の表示がでる。

❺ 「サンスウ　キョウシツ」のタイトルがでて、次の画面で終る。

```
RETURN without GOSUB in 1090
Ok
█
```

エラーメッセージが表示されて終ることは、まだこのプログラムが製作途中ですのでしかたありません。しかし疑問は、1問きり出題されないことです。2030行から2130行にまたがるFOR～NEXTループもちゃんとしています。ですから当然10回ループするはずです。

●正常にループしていないと思われるときはPRINT文でループ変数の中味を調べてみましょう。

2125行にSTOP文を入れ (2125　STOP ⏎)、もう一度RUNさせてピーッという音でBreakがかかったら、ダイレクトモードで次のようにキーインし、変数Iの値をチェックしてみましょう。

```
? I
```

●FOR～NEXTループを抜けるときループ変数の値は(最終値+1)になっています。

1001という数値がかえってきました。これはなぜかと申しますと、「アタリ」のサブルーチンを抜けでるとき、6100行で同じループ変数Iを使っていますから、このときの値が代入されてしまったのです。したがって、たし算のルーチンに戻ってもすでにIの値は10以上なので、ループを抜けだしてしまいます。次のループ変数を変えましょう。

```
6030   FOR J=1 TO 200:BEEP 1:BEEP 0:NEXT J
6100   FOR J=1 TO 1000:NEXT J
7100   FOR J=1 TO 1000:NEXT J
```

●緑色のところを変更してください。

5・8　得点サブルーチン

ループ変数ＩをＪに変えてからRUNさせてみましたか？

今度は10回出題されるようになりましたね。

そうしたら次は、１問正解したら10点として、得点を画面に表示するようにしましょう。やはりサブルーチン扱いにします。

たし算のサブルーチンから帰ったら得点サブルーチンに飛ばします。行番号を230、ラベル名を「＊TOKUTEN」として次のステートメントをキーインします。

```
230 GOSUB *TOKUTEN
```

このサブルーチンは複雑ではありませんので、まず総リストを示します。

```
3000 '
3010 *TOKUTEN
3020  TOTAL=SEIKAI*10
3030  LOCATE 8,8
3040  PRINT "ｷﾐ ﾉ ﾄｸﾃﾝ ﾊ";TOTAL;"ﾃﾝ ﾃﾞｽ"
3050  KAISU=KAISU+1:B(KAISU)=TOTAL
3060  FOR I=1 TO 2000:NEXT I
3070  CLS
3080 RETURN
```

正解数は変数SEIKAIの中に入っていますから、これを10倍すれば得点が計算できます。この得点を変数TOTALにしまいます。そして、これを3040行のPRINT文で画面に表示しています。

3050行の配列Ｂは、複数回チャレンジする人もいると思いますので、各回の得点を格納しておくものです。配列の添字は変数KAISUでカウントします。

初回の場合ですと、KAISUの中味は０の状態でこのルーチンに飛び込みますから、3050行の左辺のKAISUには、右辺の

KAISU＋１＝０＋１＝１

が代入されます。そしてこれが添字になり、Ｂ(１)に第１回目の得点が格納されることになります。２回目は、KAISUの中味が1だけ加算されますから、今度はＢ(2) に第２回目の得点がしまわれます。

3060行はタイマーですね。そして、画面をクリアしてリターンです。

5·9　もっとチャレンジしたければ

この算数教室がおもしろくてやめられない子どものために、それをフォローする部分を設けておきましょう。

得点表示から戻ってから、もう一度チャレンジするかメッセージを表示して意志を問います。

```
240 LOCATE 8,6
250 PRINT "ﾁｶﾞｳ ﾄｲ ﾆ ﾁｮｳｾﾝ ｼﾏｽｶ ?"
260 LOCATE 8,7
270 PRINT "     ﾊｲ=1  ｲｲｴ=2"
```

ここではINPUT文を使っていませんから、キー入力を察知するINKEY$を使います。

```
280 Y$=INKEY$:IF Y$="" THEN 280
290 IF Y$="1" THEN CLS:GOTO 140 ELSE IF Y$="2"
              THEN GOTO 300 ELSE GOTO 240
```

●290行はIF～THEN～ELSE文といいます。IF文の次の条件が満たされればTHEN以下が、そうでなければELSE以下が実行されます。

おや？　複雑そうな条件文がでてきましたね。しかしこれもたいしたことはありませんよ。意味はこうなります。

もしキー入力が「1」ならばメニュー画面へ、「1」ではなく「2」だったなら300行へ、「1」でも「2」でもなければキー入力ミスですから、再度、チャレンジするかどうかの意志を問います。

「2」を入力して、「もうイヤだ」と答えた人が、2回以上のチャレンジャーでしたら、総合評価をして終りです。

```
300 IF KAISU>1 THEN GOSUB *HYOKA
310 END
```

```
240 LOCATE 8,6
250 PRINT "ﾁｶﾞｳ ﾄｲ ﾆ ﾁｮｳｾﾝ ｼﾏｽｶ ?"
260 LOCATE 8,7
270 PRINT "     ﾊｲ=1  ｲｲｴ=2"
280 Y$=INKEY$:IF Y$="" THEN 280
290 IF Y$="1" THEN CLS:GOTO 140 ELSE IF Y$="2"
              THEN GOTO 300 ELSE GOTO 240
300 IF KAISU>1 THEN GOSUB *HYOKA
310 END
```

5・10　総合評価サブルーチン

数値を加算する

このルーチンでは点数評価をしますので、各回の得点を加算し、その合計を回数で割って平均点をだすようにします。そして、その平均点いかんによって、評価メッセージを出力します。

このサブルーチンをキーインする前に、加算の原理について説明します。

私たちはこれまで何度か、カウンタを使って１ずつ加算したことがありますね。この原理はもうおわかりだと思います。次のような形になります。キーインしてみてください。

```
5 FOR I=1 TO 10
6  T=T+1
7 NEXT I
8 PRINT T
9 END
```

これは、１ずつ10回カウントするプログラムです。意味がはっきりわからない方は88ページをもう一度お読みください。

(f・5)キーを押して実行してみましょう。

```
run
 10
Ok
█
```

１ずつ10回カウントするのですから答は当然10になります。

では、カーソルを移動させて行番号６の「１」を「Ｉ」に変えてみてください。修正したら必ずリターンキーを押してください。そうしないと修正したことになりません。

```
6 T=T+I
```

これが、もう少し一般化した加算形式です。Ｉはループ変数ですから、合計用の変数Ｔに1、2、3、4……と10まで次々に加えていきます。

ですから、これをRUNさせれば、そうです、１から10までの和、つまり55が得られます。

```
run
 55
Ok
█
```

それでは次のステートメントをキーインしてください。配列を使っていますが、添字は10までですので配列宣言は不要ですね。

```
1 FOR I=1 TO 10
2  B(I)=I
3 NEXT I
```

これは、配列Bの要素1から10までに、それぞれ1、2、3、……と10までの整数を格納するプログラムです。

そして次に、行番号6をまた変更します。

```
6 T=T+B(I)
```

ループ変数Iの代わりに配列変数を使いました。

それでは、画面をクリアして行番号9までのリストを表示してみましょう。

```
list -10    ←行番号10までのリストを表示
1 FOR I=1 TO 10
2  B(I)=I
3 NEXT I
5 FOR I=1 TO 10
6 T=T+B(I)
7 NEXT I
8 PRINT T
9 END
Ok
run
 55
Ok
█
```

LISTコマンドでは、マイナス記号、行番号の順にキーインすると、最初からその行までのリストが出力されます。

プログラムリストに間違いがなかったら、RUNしてみてください。1から10まで和が得られます。つまり、行番号5から7のループで、

配列の中味を次々に加算していったのです。

わかりましたか？　これが数値を合計する原理です。これさえ理解できれば、得点加算は簡単です。

行番号９までは不要ですので、削除しておきましょう。

```
delete 1-9
```

総合評価サブルーチン

得点加算についていまくわしく説明しましたので、ここもいきなりプログラムリストを示します。

```
4000 '
4010 *HYOKA
4020 CLS
4030  FOR I=1 TO KAISU
4040   G.TOTAL=G.TOTAL+B(I)
4050  NEXT I
4060  AVERAGE=G.TOTAL/KAISU
4070  LOCATE 8,8
4080  PRINT "ｷﾐ ﾉ ﾍｲｷﾝ ﾊ";AVERAGE;"ﾃﾝ ﾃﾞｽ"
4090  IF AVERAGE<50 THEN LOCATE 8,10:PRINT
      "ｽｸｲﾖｳ ｶﾞ ｱﾘﾏｾﾝ":GOTO 4130
4100  IF AVERAGE<70 THEN LOCATE 8,10:PRINT
      "ﾓｳｽｺｼ ﾍﾞﾝｷｮｳ ｼﾃｸﾀﾞｻｲ":GOTO 4130
4110  IF AVERAGE<90 THEN LOCATE 8,10:PRINT
      "ﾏｱﾏｱ ﾃﾞｽﾈ":GOTO 4130
4120  LOCATE 8,10:PRINT "ﾖｸﾃﾞｷﾏｼﾀ"
4130 RETURN
```

総合得点の計算（4030〜4040）

平均点を算出（4060）

行番号4030から4050までが総合得点を計算する部分です。ループ変数Ｉは、チャレンジした回数だけループさせればよいので、**変数KAISUの中味まで**ということになります。

配列Ｂの中に各回の得点がしまわれていますから、それらを加算し**グランドトータル用変数G.TOTALに代入**します。

平均点は総合得点を回数で割ればよいですね。行番号4060がその処理です。**結果は平均点用の変数AVERAGEにしまわれます。**

行番号4080以降は評価の画面表示です。

評価基準は50点未満（4090行）、50点以上70点未満（4100行）、70点以上90点未満（4110行）、90点以上（4120行）です。

この辺はみなさんが適当に変えてください。

5・11　またまた動作テスト

いろいろなサブルーチンを追加することによって、たし算のプログラムがほぼ完成しました。ここで動作テストをしてみましょう。

まずRUNさせますと、例によってタイトルが表示され、次にメニュー画面になります。ここでは1をキーインしてください。

するとすぐに問題がだされ、正解かどうかによって、「アタリ」「ハズレ」のメッセージが表示されます。

10問終わりますと、次のメッセージです。

```
キミ ノ トクテン ハ 70 テン デ゛ス
```

間もなくするとこれが消えて、次の画面です。

```
チカ゛ウ トイ ニ チョウセン シマスカ ？
      ハイ=1   イイエ=2
```

チャレンジすることにして、1をキーインします。すると再びメニューに戻りますから、1を入力しましょう。

またまた出題されました。すみませんが、10問答えてみてください。10問終わると、おなじみのメッセージです。

```
キミ ノ トクテン ハ 170 テン デ゛ス
```

ここに1つの問題点があります。

ごらんのように2回目の得点が100点以上になってしまいました。1問10点として計算しますので、いくらがんばっても100点をこえることはないわけです。それなのになぜ？

これは、たし算のサブルーチンに飛んだときに、「正解数」用の変数SEIKAIをゼロクリアするのを忘れてしまったためです。つまり、1回目に正解した回数がそのまま2回目にも持ちこまれてしまったのです。

2030行のFOR～NEXTループに入る前に変数をクリアしておきましょう。

```
2025  SEIKAI=0
```

今度はだいじょうぶだと思います。
2回以上やって終りにしますと、次の評価メッセージです。

```
キミ ノ ヘイキン ハ 78 テン デ゛ス

マアマア デ゛スネ
```

いつも同じ問題？

(f･5)キーで何回か実行してみるとわかると思いますが、実行するたびに同じ問題が出題されます。これではまるきりテストにはなりませんね。

これは2数を作り出すサブルーチンで使ったRND関数に原因があるのです。名前はランダムですが、RUNコマンドによって実行すると、いつもスタートラインが一定しているのです。

このことは前章でもふれましたが、もう一度考えてみましょう。

次のプログラムをキーインしてください。算数教室のプログラムはこのままでいいですよ。

```
1 R=INT(RND(1)*100)
2 PRINT R
3 '
4 RUN
```

これは、行番号1で0以上100未満のランダムな整数を1つ発生させ、それを行番号2で画面に表示します。そして、行番号4のRUNコマンドがありますから、行番号1と2の処理を何回も新規に実行します。(f･5)キーでRUNさせてみてください。

```
run
 24
 24
 24
 24
^C
Break in 2
Ok
█
```

適当なところで(STOP)キーを押して止めてください。

いつも同じ整数が表示されます。つまり、スタートラインはいつも一定しているということですね。

ですから、たし算用の2数を作りだすときも、RND関数のスタートラインを狂わしておかなければならないのです。そうしませんと、常に同じ問題が出題されてしまいます。

132ページと同じように、INKEY$を使って、乱数をかきまぜることにしました。

```
192 Y$=INKEY$:IF Y$="" THEN R=RND(1):GOTO 192
```

これで、何かキー入力があるまで乱数がかきまぜられます。

ところで、行番号190ではINPUT文でキー入力を要求しますが、今度の変更で、キー入力はINKEY$が受けもつことになりましたので、INPUT文は不要になります。緑色で示したように修正してください。

```
190 LOCATE 9,13:PRINT " ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ"
```

VAL関数

●この関数は文字型の数字を数値に変換します。

そして、入力された数字は文字列としてY$に入りますから、220行のON～GOSUB文をそのまま使うために数値に変換しておかなければなりません。これを引き受けてくれるのがVAL（バル、またはバリュー）関数です。

次のステートメントをキーインしてください。

```
194 NUMBER=VAL(Y$)
```

これでY$という文字が数値に変換され、数値変数NUMBERに入りました。

乱数をかきまぜる作業はこれで終りです。実行して確かめてみてください。

配列の宣言

行番号3050で配列Bを使っていますので、続けて11回以上くりかえすと、次のエラーになります。

```
Subscript out of range in 3050
```

万全を期して、110行で配列宣言をしておきましょう。

```
110 DEFINT A-Z:DIM B(20)
```

5·12 ひき算と九九

いよいよ、この算数教室プログラムも最終段階にきました。残る2つのテーマを解決しておしまいです。

ひき算も九九も演算記号が変わるだけですから、たし算のサブルーチンをスクリーンエディトで編集しなおすことにします。

ひき算のサブルーチン

まず、たし算のサブルーチンを画面に表示してください。

```
list 2000-2140
2000 '
2010 *TASHIZAN
2020  COLOR 4
2025  SEIKAI=0
2030  FOR I=1 TO 10
2040   LOCATE 11,7:PRINT "ﾓﾝﾀﾞｲ(";I;")"
2050   GOSUB *KAZU
2060   KOTAE=KAZU1+KAZU2
2070   LOCATE 11,9
2080   PRINT KAZU1;"+";KAZU2;" ="
2090   LOCATE 11,11:INPUT "ｺﾀｴ ﾊ ";ANSER
2100    IF ANSER=KOTAE THEN GOSUB *ATARI
      :SEIKAI=SEIKAI+1:GOTO 2120
2110   GOSUB *HAZURE
2120   CLS
2130  NEXT I
2140 RETURN
Ok
█
```

上の画面をごらんください。

ひき算の部分は、まず四角で囲ったスミアミの部分がありますから、ここを変更します。

変更後のリストを次ページに示します。緑色の部分が修正個所ですから、これをよくみながらなおしていってください。修正は各行ごとに行います。たとえば、2000行でしたら、カーソルを「2」に重ね、2 5 0 0 とキーインしリターンキーを押します。するとカーソルは次の行の先頭にきますから、同じように修正してください。

●あるサブルーチンの行番号を変更して他のサブルーチンに転用するときに注意することは、ステートメント中のGOTO文の飛び先は変更されないことです。上の2100行中のGOTO文の飛び先2120は2630に変えてください。

```
2500 '
2510 *HIKIZAN
2520  COLOR 4
2530  SEIKAI=0
2540  FOR I=1 TO 10
2550   LOCATE 11,7:PRINT "ﾓﾝﾀﾞｲ(";I;")"
2560   GOSUB *KAZU
2570   KOTAE=KAZU1-KAZU2
2580   LOCATE 11,9
2590   PRINT KAZU1;" -";KAZU2;" ="
2600   LOCATE 11,11:INPUT "ｺﾀｴ ﾊ ";ANSER
2610    IF ANSER=KOTAE THEN GOSUB *ATARI
      :SEIKAI=SEIKAI+1:GOTO 2630
2620   GOSUB *HAZURE
2630   CLS
2640  NEXT I
2650 RETURN
```

各行ごとにリターンキーを忘れないでください。

修正が終りましたら、小学低学年を対象にしますと、答がマイナスになるのはまずいので、プラスになるように調整します。次のステートメントをキーインしてください。

```
2565   IF KAZU1<KAZU2 THEN GOTO 2560
```

つまり、ひかれる数のほうが小さかったら、もう一度サブルーチンに飛ばして新しい２数を作ります。

これでひき算のサブルーチンは完成です。

九九のサブルーチン

九九の場合は、かけ合わせる２数が１から９までの数ですので、次のサブルーチンをあらかじめキーインします。

```
5500 '
5510 *SUUCHI
5520  KAZU1=INT(RND(1)*9)+1
5530  KAZU2=INT(RND(1)*9)+1
5540 RETURN
```

１～９の２数を作る

これで１桁の２数を作ることができます。

あとは、たし算のサブルーチンを使って、ひき算の場合とまったく同じ方法で、編集しなおします。

前項と同様に、変更前と変後のリストを示します。

```
list 2000-2140
2000 '
2010 *TASHIZAN
2020  COLOR 4
2025  SEIKAI=0
2030  FOR I=1 TO 10
2040   LOCATE 11,7:PRINT "ﾓﾝﾀﾞｲ(";I;")"
2050   GOSUB *KAZU
2060   KOTAE=KAZU1+KAZU2
2070   LOCATE 11,9
2080   PRINT KAZU1;"+";KAZU2;" ="
2090   LOCATE 11,11:INPUT "ｺﾀｴ ﾊ ";ANSER
2100    IF ANSER=KOTAE THEN GOSUB *ATARI
      :SEIKAI=SEIKAI+1:GOTO 2120
2110   GOSUB *HAZURE
2120   CLS
2130  NEXT I
2140 RETURN
Ok
```

```
2800 '
2810 *KUKU
2820  COLOR 4
2830  SEIKAI=0
2840  FOR I=1 TO 10
2850   LOCATE 11,7:PRINT "ﾓﾝﾀﾞｲ(";I;")"
2860   GOSUB *SUUCHI
2870   KOTAE=KAZU1*KAZU2
2880   LOCATE 11,9
2890   PRINT KAZU1;" ×";KAZU2;" ="
2900   LOCATE 11,11:INPUT "ｺﾀｴ ﾊ ";ANSER
2910    IF ANSER=KOTAE THEN GOSUB *ATARI
      :SEIKAI=SEIKAI+1:GOTO 2930
2920   GOSUB *HAZURE
2930   CLS
2940  NEXT I
2950 RETURN
```

5·13 プログラムの完成

REM文を入れる

どうもごくろうさまでした。最後の総仕上げをして終りにしましょう。まず、プログラム先頭に注釈を入れます。

```
1 ' サンスウ キョウシツ
2 ' file name ==> SANSUU
3 ' 1983.11.15 サクセイ
4 ' save "SANSUU"
5 '
```

次に、さらにプログラムを見やすくするために、次のREM文を入れておきましょう。

```
125 '
135 '
225 '
235 '
305 '
```

行番号を整理する

そして最後は**リナンバリング**です。サブルーチンは、ちょうど切りのよい行番号から始まるようにしましょう。

```
renum 100
Ok
renum 1000,440
Ok
renum 1500,1150
Ok
renum 2000,1660
Ok
renum 2500,2170
Ok
renum 3000,2660
Ok
```

```
renum 3500,3090
Ok
renum 4000,3640
Ok
renum 4500,4050
Ok
renum 5000,4550
Ok
save "SANSUU"
Ok
█
```

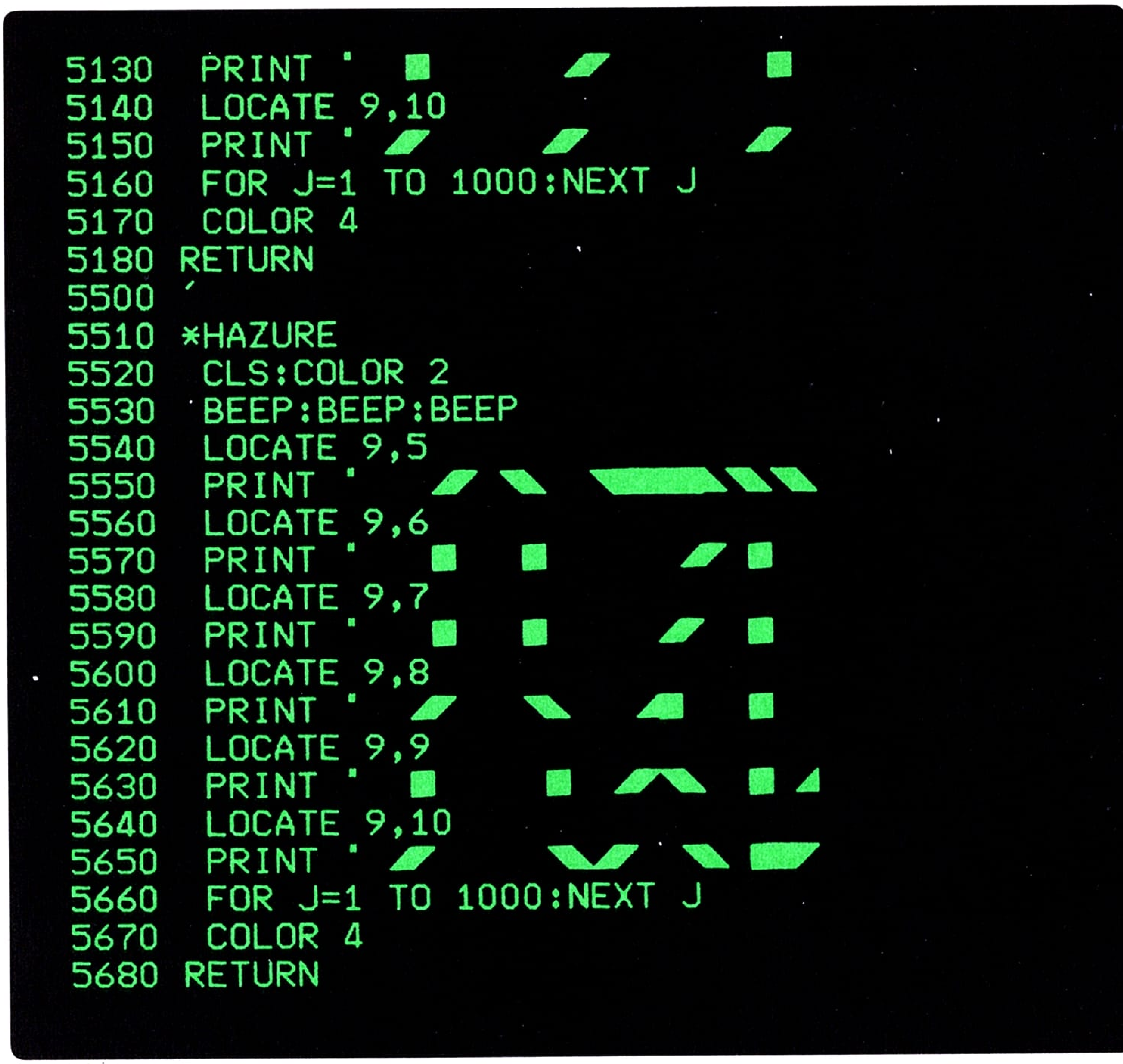
プログラムリスト

```
100 ' ｻﾝｽｳ ｷｮｳｼﾂ
110 ' file name ==> SANSUU
120 ' 1983.11.15 ｻｸｾｲ
130 ' save "SANSUU"
140 '
150 CONSOLE 0,25,0,1:WIDTH 40,25
160 DEFINT A-Z:DIM B(20)
170 CLS:COLOR 6
180 '
190 GOSUB *MIDASHI
200 '
210 LOCATE 9,7 :PRINT "ﾄﾞﾉ ﾓﾝﾀﾞｲ ｦ ﾔﾘﾏｽｶ ?"
220 LOCATE 9,9 :PRINT "  1･･･ﾀｼｻﾞﾝ"
230 LOCATE 9,10:PRINT "  2･･･ﾋｷｻﾞﾝ"
240 LOCATE 9,11:PRINT "  3･･･ｸｸ"
250 COLOR 2
260 LOCATE 9,13:PRINT " ﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ "
270 Y$=INKEY$:IF Y$="" THEN R=RND(1):GOTO 270
280 NUMBER=VAL(Y$)
290 IF NUMBER>3 THEN BEEP:COLOR 6:GOTO 210
300 CLS
310 ON NUMBER GOSUB *TASHIZAN,*HIKIZAN,*KUKU
320 '
330 GOSUB *TOKUTEN
340 '
350 LOCATE 8,6
```

```
360 PRINT "ﾁｶﾞｳ ﾄｲ ﾆ ﾁｮｳｾﾝ ｼﾏｽｶ ?"
370 LOCATE 8,7
380 PRINT "     ﾊｲ=1  ｲｲｴ=2"
390 Y$=INKEY$:IF Y$="" THEN 390
400 IF Y$="1" THEN CLS:GOTO 210 ELSE IF Y$="2"
              THEN GOTO 410 ELSE GOTO 350
410 IF KAISU>1 THEN GOSUB *HYOKA
420 '
430 END
1000 '
1010 *MIDASHI
1020  LOCATE 9,8
1030  PRINT "*------------------*"
1040  LOCATE 9,9
1050  PRINT "|                  |
1060  LOCATE 9,10
1070  PRINT "|    ｻﾝｽｳ ｷｮｳｼﾂ    |
1080  LOCATE 9,11
1090  PRINT "|                  |
1100  LOCATE 9,12
1110  PRINT "*------------------*
1120  FOR I=1 TO 5000:NEXT I
1130  CLS
1140 RETURN
1500 '
1510 *TASHIZAN
1520  COLOR 4
1530  SEIKAI=0
1540  FOR I=1 TO 10
1550   LOCATE 11,7:PRINT "ﾓﾝﾀﾞｲ(";I;")"
1560   GOSUB *KAZU
1570   KOTAE=KAZU1+KAZU2
1580   LOCATE 11,9
1590   PRINT KAZU1;"+";KAZU2;" ="
1600   LOCATE 11,11:INPUT "ｺﾀｴ ﾊ ";ANSER
1610    IF ANSER=KOTAE THEN GOSUB *ATARI
      :SEIKAI=SEIKAI+1:GOTO 1630
1620   GOSUB *HAZURE
1630   CLS
1640  NEXT I
1650 RETURN
2000 '
```

```
2010 *HIKIZAN
2020  COLOR 4
2030  SEIKAI=0
2040  FOR I=1 TO 10
2050   LOCATE 11,7:PRINT "ﾓﾝﾀﾞｲ(";I;")"
2060   GOSUB *KAZU
2070   IF KAZU1<KAZU2 THEN GOTO 2060
2080   KOTAE=KAZU1-KAZU2
2090   LOCATE 11,9
2100   PRINT KAZU1;" -";KAZU2;" ="
2110   LOCATE 11,11:INPUT "ｺﾀｴ ﾊ ";ANSER
2120    IF ANSER=KOTAE THEN GOSUB *ATARI
          :SEIKAI=SEIKAI+1:GOTO 2140
2130   GOSUB *HAZURE
2140   CLS
2150  NEXT I
2160 RETURN
2500 '
2510 *KUKU
2520  COLOR 4
2530  SEIKAI=0
2540  FOR I=1 TO 10
2550   LOCATE 11,7:PRINT "ﾓﾝﾀﾞｲ(";I;")"
2560   GOSUB *SUUCHI
2570   KOTAE=KAZU1*KAZU2
2580   LOCATE 11,9
2590   PRINT KAZU1;" x";KAZU2;" ="
2600   LOCATE 11,11:INPUT "ｺﾀｴ ﾊ ";ANSER
2610    IF ANSER=KOTAE THEN GOSUB *ATARI
          :SEIKAI=SEIKAI+1:GOTO 2630
2620   GOSUB *HAZURE
2630   CLS
2640  NEXT I
2650 RETURN
3000 '
3010 *TOKUTEN
3020  TOTAL=SEIKAI*10
3030  LOCATE 8,8
3040  PRINT "ｷﾐ ﾉ ﾄｸﾃﾝ ﾊ";TOTAL;"ﾃﾝ ﾃﾞｽ"
3050  KAISU=KAISU+1:B(KAISU)=TOTAL
3060  FOR I=1 TO 2000:NEXT I
```

```
3070  CLS
3080 RETURN
3500 '
3510 *HYOKA
3520 CLS
3530  FOR I=1 TO KAISU
3540   G.TOTAL=G.TOTAL+B(I)
3550  NEXT I
3560  AVERAGE=G.TOTAL/KAISU
3570  LOCATE 8,8
3580  PRINT "ｷﾐ ﾉ ﾍｲｷﾝ ﾊ";AVERAGE;"ﾃﾝ ﾃﾞｽ"
3590  IF AVERAGE<50 THEN LOCATE 8,10:PRINT
      "ｽｸｲﾖｳ ｶﾞ ｱﾘﾏｾﾝ":GOTO 3630
3600  IF AVERAGE<70 THEN LOCATE 8,10:PRINT
      "ﾓｳｽｺｼ ﾍﾞﾝｷｮｳ ｼﾃｸﾀﾞｻｲ":GOTO 3630
3610  IF AVERAGE<90 THEN LOCATE 8,10:PRINT
      "ﾏｱﾏｱ ﾃﾞｽﾈ":GOTO 3630
3620  LOCATE 8,10:PRINT "ﾖｸﾃﾞｷﾏｼﾀ"
3630 RETURN
4000 '
4010 *KAZU
4020  KAZU1=INT(RND(1)*100)
4030  KAZU2=INT(RND(1)*100)
4040 RETURN
4500 '
4510 *SUUCHI
4520  KAZU1=INT(RND(1)*9)+1
4530  KAZU2=INT(RND(1)*9)+1
4540 RETURN
5000 '
5010 *ATARI
5020  CLS:COLOR 1
5030  FOR J=1 TO 200:BEEP 1:BEEP 0:NEXT J
5040  LOCATE 9,5
5050  PRINT "◥■■◣ ◢■■  ◢ ■
5060  LOCATE 9,6
5070  PRINT "   ◢  ◤  ■ ■ ■
5080  LOCATE 9,7
5090  PRINT "  ■◤   ◥■◤ ■ ■
5100  LOCATE 9,8
5110  PRINT "  ■    ◢    ■
5120  LOCATE 9,9
```


5130 PRINT "
5140 LOCATE 9,10
5150 PRINT "
5160 FOR J=1 TO 1000:NEXT J
5170 COLOR 4
5180 RETURN
5500 '
5510 *HAZURE
5520 CLS:COLOR 2
5530 BEEP:BEEP:BEEP
5540 LOCATE 9,5
5550 PRINT "
5560 LOCATE 9,6
5570 PRINT "
5580 LOCATE 9,7
5590 PRINT "
5600 LOCATE 9,8
5610 PRINT "
5620 LOCATE 9,9
5630 PRINT "
5640 LOCATE 9,10
5650 PRINT "
5660 FOR J=1 TO 1000:NEXT J
5670 COLOR 4
5680 RETURN

第6章

ダイレクトメール

6・1　DMプログラム

前章までで、「プログラムとはこんなものか」ということがご理解いただけたのではないかと思います。

この章では、まだ取り上げていない「データの扱い」についてダイレクトメールプログラムを例にとって説明いたします。

DMプログラムとは、

コード，氏名，郵便番号，住所，番地

などの項目をあらかじめ登録しておいて、必要なときに必要な人への宛て名をプリンタに出力するものです。

プリンタは、あればそれに越したことはありませんが、**なくてもかまいません。**

入力方法は、宛て名書きの必要な人のコード番号または氏名をキーインすることによって行います。

また、ディスプレイ画面での宛て名表示は、おおよそ次のようなスタイルになります。

```
101
ﾁﾖﾀﾞ ｸ ｶﾝﾀﾞ ｼﾞﾝﾎﾞｳﾁｮｳ
                 1-52
  ｶﾌﾞｼｷｶｲｼｬ ﾅﾂﾒｼｬ ｻﾏ
```

これをプリンタに出力したいときは、PRINT文をLPRINT文に変更するだけですので、プリンタをお持ちの方はぜひためしてみてください。

プログラムの流れ

まず、初期設定です。次に登録してある全データを配列の中に読み込みます。そして、データをコードで探すのか氏名で探すのかメニューを表示します。メニュー選択にもとづいてデータを探し、それを一時的に配列の中に退避させておき、最後に一括してディスプレイに出力します。

6・2　初期設定とデータ読み込み

初期設定

これはもう、みなさんおなじみのステートメントばかりです。

```
10 CONSOLE 0,25,0,0:WIDTH 40,25
20 DEFINT A-Z
30 CLS
40 DIM COD(100),NAM$(100)
50 DIM PST$(100),ADR$(100),BAN$(100)
60 '
```

5つの配列が宣言してありますが、それぞれCODはコード用の、NAM$は氏名用、PST$は郵便番号、ADR$は住所、BAN$は番地用の配列です。

コードは数値ですので、配列名にドルマークは必要ありませんね。それ以外はすべて文字型配列にしなければなりません。

配列の大きさは一応100だけ確保してありますが、これは100人分ということですから、それ以上のデータを扱う人はこの数値を変更してください。

データの読み込み

さて、データを配列内に読み込む方法ですが、**DATA（データ）文**と**READ（リード）文**の組み合わせにより行います。　DATA

次のプログラムをキーインしてみてください。　READ

```
1 RESTORE 5          ←読み出すデータの位置を設定します
2 READ A%,B,C$       ←データを変数の中に読み込みます
3 PRINT A%;"  ";B!;"  ";C$   ←読んだデータを表示します
4 '
5 DATA 12345,123456,aa
6 STOP
```

このプログラムがデータ読み込みの基本的パターンです。

行番号1の**RESTORE（リストア）文**は、うしろに行番号をつけて読み込みデータの位置を指定します。行番号を省略すると、そのプログラムの1番最初のデータに読み込み位置が設定されます。　RESTORE

行番号２のREAD文で行番号５のデータを順番に読み、変数Ａ％、Ｂ、Ｃ＄にとりこみます。

そして、行番号３で、いま読み込んだデータを画面に表示します。

実行させてみてください。画面は次のようになりましたか？

```
run
 12345     123456    aa
Break in 6
Ok
█
```

整数型変数Ａ％に12345が、ここではイクスクラメーションマークをつけていませんが単精度型変数Ｂに123456が、文字型変数にaaが格納されたのがわかりますね。

では、行番号５を次のように変更してみてください。

```
5 DATA 123456,123456,aa
```

最初のデータに「６」をつけ加えました。つまり単精度型になったわけです。これでRUNさせてみてください。

次のようなエラーメッセージがでてしまいました。

```
run
Overflow in 2
Ok
█
```

これはなぜかといいますと、整数型変数Ａ％に整数の範囲を超える単精度型実数を読み込もうとしたためです。2·5節でお話ししましたね。整数型はマンションにたとえると２ＤＫ、単精度型は４ＤＫです。ですから、全部入りきらないであふれてしまった、ということです。

このように、読み込み用の変数とデータの型は一致していなければなりません。

今度は、行番号５を次のように変えてみてください。最初のデータは整数型になおし、３つ目のデータを削除してしまいました。

```
5 DATA 12345,123456
```

これを実行するとどうなるでしょうか？

(f･5) キーを押してください。

```
run
Out of DATA in 2
Ok
█
```

またエラーメッセージです。意味は「データがたりない」ということです。

つまり、A%、Bに対応するデータは読み込みましたが、C$に対応するデータが不足していたわけです。

行番号5のDATA文はコンマで区切ってありますね。コンピュータは、このコンマごとにデータを読み込んでいきますので、もしコンマを1つつけ忘れたりしますと、つけ忘れたコンマの両はじのデータは1つと考えられてしまいますので、最終的にはデータ量が1個少なくなってしまい、同じエラーが生ずることがあります。

さあ、これでデータの読み込みは理解していただけたと思います。DMプログラムのデータを入力しましょう。その前に、行番号6までは不要ですので削除しておきましょう。

```
delete 1-6 ⏎
```

データは、

コード，氏名，郵便番号，住所，番地

の順で入力します。コンマを忘れないようにしてください。

```
9000 '
9010 DATA 1,ｱｸﾀｶﾞﾜ ﾘｭｳﾉｽｹ,110,AAAAAAAA,aaa
9020 DATA 2,ﾓﾘ ｵｳｶﾞｲ,101,BBBBBBBB,bbb
9030 DATA 3,ﾅﾂﾒ ｿｳｾｷ,100,CCCCCCCC,ccc
9040 DATA 4,ｼﾏｻﾞｷ ﾄｳｿﾝ,231,DDDDDDDD,ddd
9050 DATA 5,ｸﾆｷﾀﾞ ﾄﾞｯﾎﾟ,201,EEEEEEEE,eee
9060 DATA 6,ﾓﾘ ﾖｼﾅｵ,300,FFFFFFFF,fff
9070 DATA 999
```

住所、番地は適当に変えてください。ここでは仮りにアルファベットを使っておきました。

行番号9070の「999」は、データの終りを識別するためのものです。文字型データの場合はよく「END」などを使いますが、この場合はコードに対応するデータですから数値でなくてはいけませんので、この

ようにしました。

また、DATA文はプログラムのうしろのほうにおいたほうが便利ですので、9000行以降としました。

次にデータ読み込み部分ですが、これはサブルーチンにしますので、次のステートメントをキーインします。

```
70 GOSUB *DAT.READ
80 '
```

そして、サブルーチンです。

```
1000 '
1010 *DAT.READ                ← ラベル
1020 RESTORE 9010             ← データ頭出し
1030 C=C+1                    ← カウンタ
1040 READ COD(C)              ← コードを読み込む
1050 IF COD(C)=999 THEN RETURN ← 終りなら戻る
1060 READ NAM$(C),PST$(C),ADR$(C),BAN$(C) ← 各データを配列に格納
1070 GOTO 1030
```

1020行のRESTORE文でデータの頭出しをします、そして、1030行でカウンタを１つあげて、1040行のREAD文でコードに対するデータを配列に読み込みます。

●ＲＵＮさせたときは変数Ｃの中味は０ですから、最初1030行では
右辺＝Ｃ＋１＝０＋１＝１となって、この１が左辺のＣに入ります。したがってカウンタは１ということになります。

読み込んだデータが「999」なら、これはデータ末尾のマークですから、サブルーチンから戻ります。

コード用配列CODに「999」が入らないかぎり、1060行のREAD文でデータをそれぞれの配列に読み込みます。

では、(f･5)キーを押してみてください*。画面がクリアされてOkが表示されました。データ読み込みが終了したのです。間違いなく読み込めたかどうかテストしてみましょう。一番最後の「999」が入っているはずの配列COD(７)の中味を表示させてみます。ダイレクトでキーインしてみてください。

＊行番号100としてＥＮＤ文を入れてからＲＵＮさせてください。
100　ＥＮＤ

```
Ok
? cod(7)   ← 間違いなさそうですね
 999
Ok
█
```

6・3 メインルーチン

この部分ではまず、希望のデータを探し出すのに、コード入力によるか、氏名入力によるかのメニューを表示します。

```
100 *MAIN
110 PRINT "ト゛チラ テ゛スカ ?"
120 PRINT "   1・・・コート゛ テ゛ サカ゛ス"
130 PRINT "   2・・・ナマエ テ゛ サカ゛ス"
140 PRINT
150 INPUT "ハ゛ンコ゛ウ ヲ ト゛ウソ゛";NUMBER
```

これはもう説明しなくてもわかりますね。

なお、ここでおことわりしておきますが、このプログラムではLOCATE文はすべて省略しましたので、PRINT文による表示は画面の左側からになります。このプログラムの完成後、みなさんが各自でLOCATE文を入れて、レイアウトを考えてみてください。

次はメニュー入力に対するそれぞれの分岐です。

```
160 IF NUMBER>2 THEN BEEP : BEEP : GOTO 110
170 IF NUMBER=1 THEN GOSUB *CODE : GOTO 190
180 GOSUB *NAMAE
190 GOSUB *DAT.PRINT
```

行番号160は、メニュー番号入力ミスに対するキャンセル処理です。

1番を選択すればコードによるデータサーチサブルーチンへ飛びます。そして戻ってきて、行番号190へ行き、データ表示のサブルーチンへジャンプします。

2番をキー入力した場合は、行番号170にひっかかりませんから、行番号180の名前によるデータサーチサブルーチンへ飛び、戻ってきて画面表示処理をします。

いずれの場合も、データの表示またはプリンタへの出力が終わったら、DMの仕事は終わりですので、次のステートメントをつけ加えておきます。

```
200 CLS
210 PRINT "DM ノ シコ゛ト ヲ オワリマス"
220 END
```

6·4 データサーチ

ここでは、メニュー番号１のコードによるデータサーチサブルーチンを作ります。

その手順は、まずＤＭを発送する人のコード番号を入力させ、それをもとにコード用の配列CODの各要素と比較し、入力コードと等しいものがあったら、そのコードに対する氏名を画面に表示させます。

そして、確認のためのメッセージを表示し、間違いなければ、そのコード番号を配列を用意し、その中にしまいます。ここまでが１プロセスで、DM発送の人数分だけこれをくりかえします。

それでは、プログラムリストを次に示します。

```
2000 '
2010 *CODE
2020 CLS
2030 INPUT "ｺｰﾄﾞ <ｵﾜﾘ=999>";CODE
2040 IF CODE=999 THEN RETURN
2050 IF CODE>C-1 THEN BEEP : GOTO 2020
2060 FOR I=1 TO C-1
2070   IF COD(I)=CODE THEN GOSUB *DAT.DISPLAY
2080 NEXT I
2090 GOTO 2020
```

データサーチ（2060～2080）

入力を終わりたいときには2030行で「999」をキーインするようにしてあります。

行番号2050行は入力ミスに対するキャンセルです。チェックポイントを「Ｃ－１」としてあるのは、178ページのデータ読み込みのルーチンで「999」を読み込んだときデータエンドと判定していますから、カウンタＣには、このときの順位が入っています。したがって、実際のコードは、ＩからＣの中味の１つ手前まででなければいけません。

入力ミスのときはブザー音を鳴らして再度入力です。

2060行から2080行がデータサーチです。

原理は、入力コードと登録してあるコードをすべて比較し、もし等しいものがあればサーチできたことになりますから、2070行のTHEN文以下で確認のためのサブルーチンへ飛ばしています。

データサーチのFOR～NEXTループを抜け出たら、再びコード番号を入力します。

データの確認

データサーチでは、データを探しだした場合、間違いがないかどうかチェックしなければなりません。それがこれから作るサブルーチンです。

まず、そのコードに対応する氏名を画面に表示します。

```
4000 '
4010 *DAT.DISPLAY
4020 CLS
4030 PRINT
4040 PRINT NAM$(I)
4050 PRINT ←————空白行を1行つくります
4060 PRINT "   トウロク シマスカ? < y/n >"
```

データサーチサブルーチンの2070行で、該当するコードが見つかったところで、このサブルーチンに飛んできたわけですから、現在変数Iにはそのコード番号がしまわれています。したがって、それに対応する氏名はNAM$(I)に入っていることになります。4040行がその画面表示です。

4060行は、このデータが必要かどうか問い正しています。この質問にはINKEY$を使って、次のようにキーインします。

```
4070 Y$=INKEY$ : IF Y$="" THEN GOTO 4070
4080 IF Y$="y" OR Y$="Y" THEN TOROKU=TOROKU+1
     : A(TOROKU)=I : RETURN
4090 IF Y$="n" OR Y$="N" THEN RETURN ELSE BEEP
     : PRINT "???" : GOTO 4050
```

論理演算子OR(オア)を使って、大文字でも小文字でも動作するようにしました。ORは「または」くらいの意味で、ORで結ばれた式の少なくとも一方が成立すればTHEN以降のステートメントを実行することになります。 OR

このデータが必要な場合には、一時的に配列にそのコード番号だけをしまっておく(登録しておく)ことにします。4080行が、その処理です。

登録用の変数TOROKUを1だけカウントし、配列Aの中にコード番号を代入します。初回の登録のときは、変数TOROKUの中味は0でここへやってきますから、1つカウントされることにより中味は1に変わります。そしてA(1)にコードが格納されることになります。2回目はA(2)に、ということになりますね。

4060行の問いかけに対してのキー入力が正しくなかった場合、4090行のELSE文以下の処理をします。

さて、4080行で新しい配列Aを使いましたので、この配列宣言をしておきましょう。次のステートメントをキーインしてください。

```
55 DIM A(100)
```

名前入力によるデータサーチ

INSTR

この場合も、コードによるデータサーチと原理は同じですが、探しだす手段としてINSTR（インストリング）関数を使います。

この関数は、文字列の中から、指定した文字列を探しだし、その位置を与えるものです。書式は次のようになっています。

INSTR（文字列１，文字列２）

この場合、文字列１の中から文字列２を探しだし、もしあればその位置を示す数値を、なければ０を返します。

次のダイレクトモードをテストしてみてください。

```
a$="abcdefg"
Ok
i=instr(a$,"fg") : h=instr(a$,"k")
Ok
? i,k
 6             0
Ok
█
```

カッコ内の文字列の表記ですが、文字型（ストリング）変数以外は必ずダブルクォーテーションでくくってください。

上の画面の場合、「fg」はＡ＄の中の６字目から入っていましたので６が、「ｋ」は見当たりませんでしたので０が返されました。

さて、このINSTR関数を使って、名前入力によるデータサーチを完成させましょう。

次ページにリストを示します。

```
3000 '
3010 *NAMAE
3020 CLS
3030 INPUT "ナマエ <オワリ=999>";NM$
3040 IF NM$="999" THEN RETURN
3050 FOR I=1 TO C-1
3060   IF INSTR(NAM$(I),NM$) THEN GOSUB
         *DAT.DISPLAY
3070 NEXT I
3080 GOTO 3020
```

行番号3060でINSTR関数を使ってデータサーチしています。これは条件文になっていますので、もし入力した名前が見つかれば0以外の数値が与えられますから、「真」と判定し、THEN文以下を実行します。

入力する名前は何文字でもかまいません。ただし、浅見さん、安藤さん、安立さんという人が登録されている中から、安立さんを探したい場合「ア」を入力したとしますと、この3人の名前にはすべて「ア」が含まれていますので、いずれの場合もチェックするべく確認のサブルーチンへジャンプしますので、それだけ手間がかかります。

プログラムの実行

それでは、ここまでのプログラムを実行して動作確認をしてみましょう。(f・5)キーを押します。

すると次のようなメニュー画面が現れます。

```
ト゛チラ テ゛スカ ?
  1・・・コート゛ テ゛ サカ゛ス
  2・・・ナマエ テ゛ サカ゛ス

ハ゛ンコ゛ウ ヲ ト゛ウソ゛? ■
```

たとえば、1をキーインしてリターンキーを押すと、次のようにコード入力の要求があります。

```
コート゛ <オワリ=999>?
```

ここで、コードを入力すればデータサーチをして、それが見つかれば画

面に表示して、登録するかどうかたずねてきます。
次の画面は1を入力した場合です。

```
アクタカ゛ワ リュウノスケ

  トウロク シマスカ? < y/n >
```

「y」でも「n」でもどちらでもかまいません。入力してみてください。すると、再びコード番号入力の画面になります。
終りたければ、「999」を入力しましょう。次のエラーメッセージが表示されて停止します。

```
コート゛ <オワリ=999>? 999
undefined label in 190
Ok
█
```

これは、行番190のGOSUB文の飛び先をまだ作っていなかったからです。

データが見つからないときは?

データサーチで指定のデータが見つからない場合、注意をうながす意味でメッセージを表示するようにしましょう。コードで入力する場合は2050行でチェックしていますので、名前で入力するケースだけを考えます。

フラグ

データサーチ後、4080行で登録してはじめてデータが見つかったということにし、それ以外を見つからなかったことにします。そしてこのことを判定するためにFOUNDという変数(フラグといいます)を用意し、登録すればFOUND=1に、そうでなければそのままに(つまりFOUND=0です)しておきます。
まず、4080行に次の緑色の部分を挿入します。

```
4080 IF Y$="y" OR Y$="Y" THEN TOROKU=TOROKU+1
     : A(TOROKU)=I : FOUND=1 : RETURN
```

ですから、名前でサーチする場合、3050行から3070行のFOR~NEXTプープを抜けでたときに、FOUNDという変数の中味をチェックし、もし0だったら、「トウロク　シテアリマセン!!」とメッセージを表示するようにします。

次のステートメントを追加しましょう。

```
3072 IF FOUND=1 THEN FOUND=0 : GOTO 3020
3074 BEEP : PRINT "トウロク シテアリマセン !!"
3076 FOR I=1 TO 2000 : NEXT I
```

3070行で、変数FOUND＝１であったらデータは見つかったわけですから、3020行へ飛ばす前にフラグ用変数をクリアしておかなければなりません。このプログラムでは間違いにならないと思いますが、フラグ用変数はその役割がすんだらクリアする習慣をつけておいたほうがよいと思うからです。

3076行のタイマーを入れておきませんと、ブザー音はしますが、メッセージは表示されたのかされないのかわかりません。

さあ、これで実行してみてください。

ためしに、DATA文にない名前を入れてみてください。次のようになりましたか。

```
ナマエ <オワリ=999>? イトウ
トウロク シテアリマセン !!
```

修正後の名前によるデータサーチサブルーチンを次に示します。

```
3000 '
3010 *NAMAE
3020 CLS
3030 INPUT "ナマエ <オワリ=999>";NM$
3040 IF NM$="999" THEN RETURN
3050 FOR I=1 TO C-1
3060   IF INSTR(NAM$(I),NM$) THEN GOSUB
         *DAT.DISPLAY
3070 NEXT I
3072 IF FOUND=1 THEN FOUND=0 : GOTO 3020
3074 BEEP : PRINT "トウロク シテアリマセン !!"
3076 FOR I=1 TO 2000 : NEXT I
3080 GOTO 3020
```

6・5 宛て名の表示

さて、残るは宛て名表示のサブルーチンです。

表示するまでの過程ですが、これは、登録したコードが配列Aにしまわれていますから、これに対応した名前、郵便番号、住所、番地をPRINT文で画面に表示するようにすればよいのです。

プログラムリストを示しますからキーインしてください。

```
5000 '
5010 *DAT.PRINT
5020 CLS
5030 FOR I=1 TO TOROKU
5040 PRINT PST$(A(I))
5050 PRINT ADR$(A(I))
5060 PRINT TAB(10);BAN$(A(I))
5070 PRINT
5080 PRINT TAB(3);NAM$(A(I));" ｻﾏ"
5090 NEXT I
5100 RETURN
```

登録した件数は変数TOROKUの中にしまわれていますから、ループ変数Iは1からこの数まで変化させればよいことになります。

次に、画面に表示する項目用の配列の添字は、配列Aに格納しておきましたから、これを充当します。

TAB

5060行、5080行に**TAB（タブ）関数**がありますが、これは画面左端からカッコ内の数だけスペースを入れるものです。たとえば5060行は、10個スペースが入って、そのうしろに番地が表示されます。

さあ、これでRUNしてみてください。

2、3人分を登録して「999」を入力すると、このサブルーチンに飛んできて宛て名を表示するはずですが、あっというまに次の画面で終ってしまいます。

```
DM ﾉ ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ
Ok
■
```

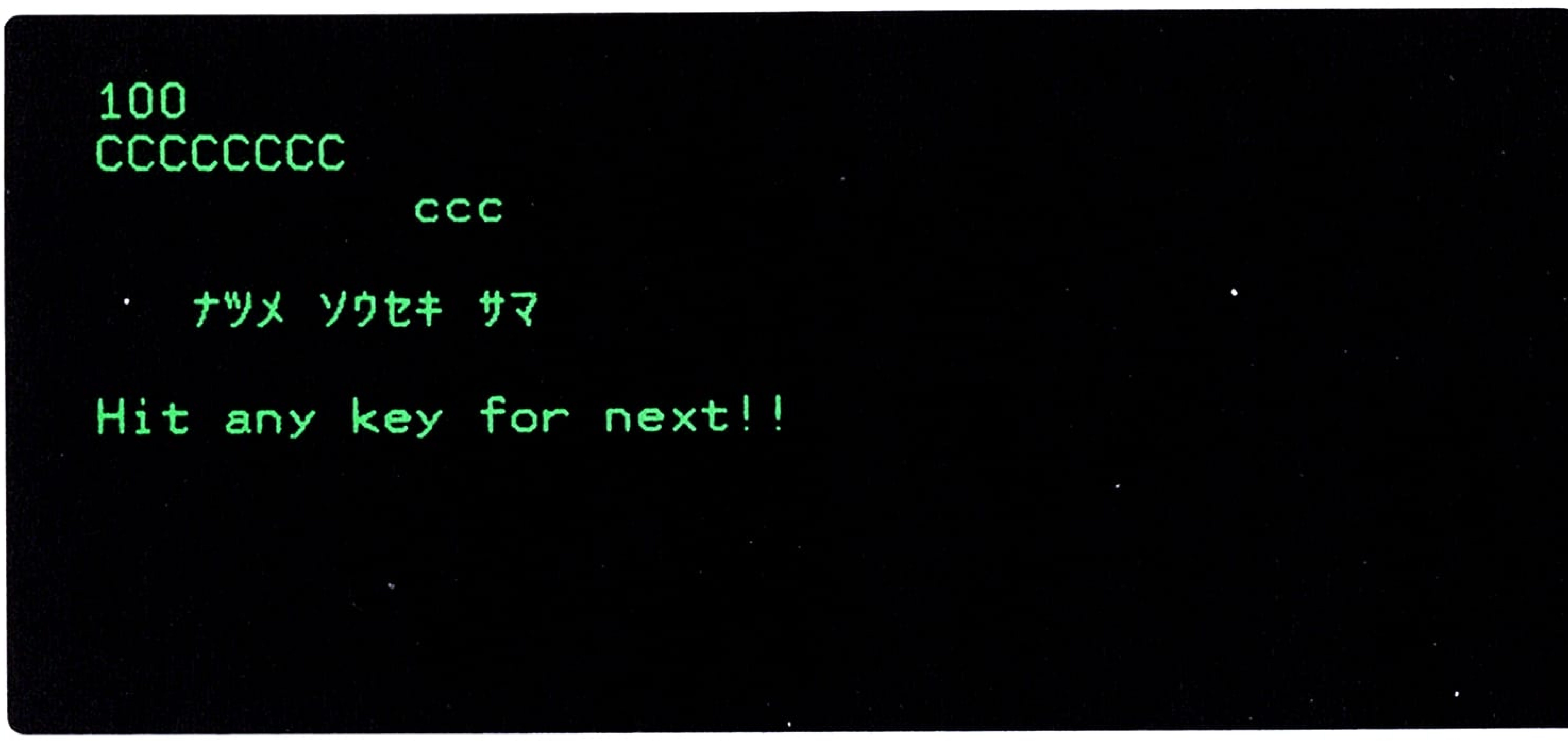
どうしてかわかりますか。

そうです。タイマーが入れてなかったのですね。

INKEY $を使って、何かキー入力があるまで、宛て名を画面に表示しておくようにしましょう。次のステートメントをキーインしてください。

```
5082 PRINT : PRINT "Hit any key for next!!"
5083 Y$=INKEY$ : IF Y$="" THEN GOTO 5083
5084 CLS
```

今度はうまくいくでしょう。RUNさせてみてください。
次のようになりました。



ここで、どのキーでも押してみてください。次の人の宛て名が表示されます。

プリンタに出力するには

プリンタへ出力したい人は5040行から5080行までの「PRINT」の前に「L」をつけて「LPRINT」としてください。
5082行のPRINT文はそのままでいいのですよ。
RUNすると、画面と同じ形で印刷されます。

なお、プリンタへの出力は、このままですとどの宛て名も続いて出力されますので、間を少しあけたいときは「LPRINT」ステートメントをいくつか入れてください。

6・6 プログラムリスト

リナンバリング

さあ、ダイレクトメールプログラムが完成しました。注釈文を入れて、リナンバリングしておきましょう。

```
1 ' ﾀﾞｲﾚｸﾄ ﾒｰﾙ
2 ' file name ==> DMAIL
3 ' 1983.11.20 ｻｸｾｲ
4 ' save "DMAIL"
5 '
```

```
renum 100
Ok
renum 1000,370
Ok
renum 2000,1080
Ok
renum 3000,2100
Ok
renum 4000,3120
Ok
renum 5000,4100
Ok
renum 6000,5140
Ok
█
```

プログラムリスト

リナンバリング後のプログラムリストを次に示します。

```
100 ' ﾀﾞｲﾚｸﾄ ﾒｰﾙ
110 ' file name ==> DMAIL
120 ' 1983.11.20 ｻｸｾｲ
130 ' save "DMAIL"
140 '
150 CONSOLE 0,25,0,0:WIDTH 40,25
```

```
160 DEFINT A-Z
170 CLS
180 DIM COD(100),NAM$(100)
190 DIM PST$(100),ADR$(100),BAN$(100)
200 DIM A(100)
210 '
220 GOSUB *DAT.READ
230 '
240 *MAIN
250 PRINT "ト゛チラ テ゛スカ ?"
260 PRINT "  1･･･コート゛ テ゛ サカ゛ス"
270 PRINT "  2･･･ナマエ テ゛ サカ゛ス"
280 PRINT
290 INPUT "ハ゛ンコ゛ウ ヲ ト゛ウソ゛";NUMBER
300 IF NUMBER>2 THEN BEEP : BEEP : GOTO 250
310 IF NUMBER=1 THEN GOSUB *CODE : GOTO 330
320 GOSUB *NAMAE
330 GOSUB *DAT.PRINT
340 CLS
350 PRINT "DM ノ シコ゛ト ヲ オワリマス"
360 END
1000 '
1010 *DAT.READ
1020 RESTORE 6010
1030 C=C+1
1040 READ COD(C)
1050 IF COD(C)=999 THEN RETURN
1060 READ NAM$(C),PST$(C),ADR$(C),BAN$(C)
1070 GOTO 1030
2000 '
2010 *CODE
2020 CLS
2030 INPUT "コート゛ <オワリ=999>";CODE
2040 IF CODE=999 THEN RETURN
2050 IF CODE>C-1 THEN BEEP : GOTO 2020
2060 FOR I=1 TO C-1
2070   IF COD(I)=CODE THEN GOSUB *DAT.DISPLAY
2080 NEXT I
2090 GOTO 2020
3000 '
3010 *NAMAE
3020 CLS
3030 INPUT "ナマエ <オワリ=999>";NM$
```

```
3040 IF NM$="999" THEN RETURN
3050 FOR I=1 TO C-1
3060   IF INSTR(NAM$(I),NM$) THEN GOSUB
         *DAT.DISPLAY
3070 NEXT I
3080 IF FOUND=1 THEN FOUND=0 : GOTO 3020
3090 BEEP : PRINT "ﾄｳﾛｸ ｼﾃｱﾘﾏｾﾝ !!"
3100 FOR I=1 TO 2000 : NEXT I
3110 GOTO 3020
4000 '
4010 *DAT.DISPLAY
4020 CLS
4030 PRINT
4040 PRINT NAM$(I)
4050 PRINT
4060 PRINT "  ﾄｳﾛｸ ｼﾏｽｶ? < y/n >"
4070 Y$=INKEY$ : IF Y$="" THEN GOTO 4070
4080 IF Y$="y" OR Y$="Y" THEN TOROKU=TOROKU+1
      : A(TOROKU)=I : FOUND=1 : RETURN
4090 IF Y$="n" OR Y$="N" THEN RETURN ELSE BEEP
      : PRINT "???" : GOTO 4050
5000 '
5010 *DAT.PRINT
5020 CLS
5030 FOR I=1 TO TOROKU
5040 PRINT PST$(A(I))
5050 PRINT ADR$(A(I))
5060 PRINT TAB(10);BAN$(A(I))
5070 PRINT
5080 PRINT TAB(3);NAM$(A(I));" ｻﾏ"
5090 PRINT : PRINT "Hit any key for next!!"
5100 Y$=INKEY$ : IF Y$="" THEN GOTO 5100
5110 CLS
5120 NEXT I
5130 RETURN
6000 '
6010 DATA 1,ｱｸﾀｶﾞﾜ ﾘｭｳﾉｽｹ,110,AAAAAAAA,aaa
6020 DATA 2,ﾓﾘ ｵｳｶﾞｲ,101,BBBBBBBB,bbb
6030 DATA 3,ﾅﾂﾒ ｿｳｾｷ,100,CCCCCCCC,ccc
6040 DATA 4,ｼﾏｻﾞｷ ﾄｳｿﾝ,231,DDDDDDDD,ddd
6050 DATA 5,ｸﾆｷﾀﾞ ﾄﾞｯﾎﾟ,201,EEEEEEEE,eee
6060 DATA 6,ﾓﾘ ﾖｼﾅｵ,300,FFFFFFFF,fff
6070 DATA 999
```

第7章

おおミステーク!

エラーメッセージ

コンピュータを習いたての頃は、ピーッという音とともに画面に出現するエラーメッセージによく出会うものです。みなさんも、本書を読みながら操作してみて、ずいぶんこの音を聞いたことと思います。

そこで、この章では、エラーメッセージが出たときどうしたらよいかを整理する意味で、ひん度の高いものについて、簡単なプログラムを例に解説したいと思います。

バグ

デバッギング

エラーは出ないが正しい結果が得られない場合、プログラム中のどこかにミスがあるわけですが、このミスのことを**バグ**（虫という意味です）といいます。そして、このバグを見つけて修正し、プログラムが正しく動作するようにすることを**デバッギング**といいます。

およそ、このデバッギングなしに、あるいはエラーメッセージに出会わずに一発でプログラムを完成させる人なんていないのです。

バグがあるのも、エラーメッセージが表示されるのも、みなプログラミング上達への1つの過程です。

そう思えば、エラーメッセージもありがたいもの。コイツがあらわれたら、「おお、ミステーク!!」とでもいって感謝しましょう。

それから、デバッギングもクイズをとくようなもので、案外楽しいものですよ。この、虫とりに生きがいを感じている人さえいる、というウワサもあるくらいですから……。

ではさっそく始めますが、画面に表示するエラーメッセージはディスクバージョンでの表記を採用しました。ディスクを使わない場合は本文中の見出しのカッコ内のメッセージが表示されるはずです。意味は変わりませんからご心配なく……。

Syntax error (?SN Error)

これはもっともよく出会うエラーメッセージです。次をキーインしてRUNさせてみましょう。

```
10 console0,25,0,0 : width72,25
run
Syntax error in 10
Ok

10 CONSOLE0,25,0,0 : WIDTH72,25
```

エラーメッセージが表示されて、行番号10に間違いがあることが指摘されています。そしてカーソルは、行頭でブリンクしています。

このエラーの意味は、構文上のエラーで「コマンド、ステートメントが定められた書式に合っていませんよ」ということです。

この場合、CONSOLE、WIDTHと、そのうしろに続くパラメータとの間にスペースが入っていないからです。

次のようにスペースを入れてなおしておきましょう。

```
10 CONSOLE 0,25,0,0 : WIDTH 72,25
```

スペースを入れる

修正してもリターンキーを押さなければいけませんよ。そうしないと修正したことにはなりませんから、またエラーがでてしまいます。

Illegal function call (?FC Error)

修正しましたら、もう一度(f･5)キーを押してみてください。画面は次のようになりましたか。

```
run
Illegal function call in 10
Ok
█
```

このエラーの意味は「指定したパラメータが許される条件を超えていますよ」ということです。

上の例では、WIDTH文の「72」が条件にそぐいません。

ＰＣ－8001では、横36桁も横72桁も許されていましたので、そのときのクセでパラメータに72を指定するとエラーになります。

また、LOCATE文で座標がマイナスになったり、COLOR文で第１パラメータにカラーコードにない「９」などを指定すると同じエラーになります。

LOCATE文などでは座標指定を数値ではなく変数を使うケースが多いですから、そのようなときこのエラーがでたら、PRINT文で座標用変数の中味を調べてみるとよいでしょう。

では、横40桁のモードに修正して次へ進みましょう。

```
10 CONSOLE 0,25,0,0 : WIDTH 40,25
```

40桁にする

ここで、例によって変数の整数宣言、画面クリアのステートメントを入れておきます。

```
20 DEFINT A-Z : CLS
```

いま私たちが入力しているプログラムも、なんらかのテーマにそって説明したほうがわかりやすいと思いますので、仮りに、ある日の商品別の売り上げ金額を計算し、それを画面に表示する、ということにしておきましょう。

?Redo from start

まず、その当日の日付を入力します。ＩＮＰＵＴ文を使えばよいですね。

```
30 INPUT YEAR$;MONTH$;DATE$
```

これでRUNさせてみましょう。

クエッションマークが表示されますから、年月日をそれぞれ2桁でコンマで区切って入力し、リターンキーを押します。すると次の画面です。

```
? 58,10,20
Syntax error in 30
Ok

30 INPUT YEAR$;MONTH$;DATE$
```

またシンタックスエラーです。カーソルは「Y」のところでブリンクしています。これ以下に間違いがありそうです。

そうです。INPUT文の変数の並びはセミコロンではなくコンマで区切るのです。

次のようにセミコロンをコンマに訂正しましょう。

```
30 INPUT YEAR$,MONTH$,DATE$
```

そしてRUNです。

```
? 58
?Redo from start
? █
```

画面に

```
?Redo from start
```

と表示されています。

これは、データを3つ入力しなければならないのに、1つきり入力しなかったからです。「もう一度最初からやりなおしなさい」ということですね。

データ数がたりなかったのですから、今度はまともに3つ入力してみましょう。

ありゃりゃ？　またまたシンタックスエラーです。

```
? 59,01,15
Syntax error in 30
Ok

30 INPUT YEAR$,MONTH$,DATE$
```

今度は、書式はちゃんとあっているのに、なぜエラーが表示されたのでしょう。

これは、使うことができない変数を使ったがためのミスです。この、使用を許可されない変数に予約語があります。 予約語

予約語とはBASICで使われているコマンドとかステートメントのことで、RUNとかPRINTなどがそれに当たります。

上の例ではDATE $が予約語だったのです。これをDAY $になおして実行してみてください。

```
30 INPUT YEAR$,MONTH$,DAY$
```

Subscript out of range (?BS Error)

次は商品名を配列に読み込む部分をプログラミングします。商品名は便宜上アルファベットを使いました。

まずDATA文をキーインしましょう。

```
1000 '
1010 DATA AA,BB,CC,DDEE
1020 DATA FF,GG,HH,II,JJ
1030 DATA KK
```

次はデータ読み込み部分です。

```
40 '
50 FOR I=1 TO 11
60   READ SHOHIN$(I)
70 NEXT I
```

さあ、実行してみましょう。

```
? 59,01,15
Subscript out of range in 60
Ok
■
```

はい、新しいエラーメッセージの登場です。

これは「配列の添字の数が制限オーバー」ということです。

最初に配列宣言をしませんでしたから、添字は10までしか許されません。それなのに、商品は11品目ありますので、ループ変数Ｉを11まで変化させていたからです。

配列宣言文を入れておきましょう。

```
15 DIM SHOHIN$(20)
```

Out of DATA (?OD Error)

もう一度RUNさせてください。

今度は、「DATA文のデータ数がたりませんよ」というエラーメッセージです。

```
? 59,1,20
Out of DATA in 60
Ok
■
```

エラー発生の行番号は「60」になっています。データ数がたりないのですからDATA文の最終行番号「1030」でもいいような気がしますが、これは、行番号60で配列SHOHIN$ (11)にデータを読み込もうとしたが不足していた、ということでエラー行番号は「60」が指し示されます。しかし、原因はDATA文のほうにありますから、チェックするのは1000行以降ということになります。

それでは、DATA文のどこが間違えているかと申しますと、1010行の「DD」と「EE」の間のコンマが落ちていたのです。

データ数がこのくらい少なければすぐわかりますが、300とか400個くらいの数になりますと、意外とコンマをつけ忘れたりするもので、たびたびこのエラーに出会います。まあ、こんなときは、気長にチェックするしかありませんね。

1010行を次のようになおしましょう。

```
1010 DATA AA,BB,CC,DD,EE
```

（「DD」と「EE」の間に）コンマを入れる

実行してみてください。

「Ok」が表示されて、データの読み込みは無事終了しました。

ここで、いまとは反対の場合、つまり「データが多すぎる場合」を検討しておきましょう。

データ数が多すぎるときは、データ不足のエラーメッセージは出ませんが、各配列と商品名がうまく対応していない場合が多々あります。

たとえば、DATA文を次のように打ち込んだとしましょう。

```
1000 '
1010 DATA AA,BB,CC,DD,EE,EE
1020 DATA FF,GG,GG,HH,II,JJ
1030 DATA KK
```

見るとわかるように、商品「EE」と「GG」を余計に打ち込んでしまいました。

これで実行してみてください。「Ok」の表示がでて何も起こりません。しかし、次のようにダイレクトにキーインしてみてください。一番最後の配列の中味をチェックするのです。

```
? shohin$(11)
II
Ok
█
```

どうですか？

11番目の商品は「KK」ですから、「II」ではなく「KK」が表示されなければなりません。配列と商品名が１対１対応をしていないのです。つまり、DATA文に何か不備があったのです。

このように、データ数が多すぎるときは何も起こりませんが、一番

うしろのデータをPRINT文でチェックしてみる必要があるのです。

DATA文をもとどおりになおしてからRUNさせ、次のようにチェックしてみてください。

```
? shohin$(11)
KK
Ok
█
```

NEXT without FOR (?NF Error)

では、次のステートメントを入力しましょう。数量と単価を入力し、金額を計算する部分です。

```
80 '
90 FOR I=1 TO 11
100 INPUT "スウリョウ,キンガク";SU,TA
110 KINGAKU$(I)=SU*TA
120 NEXT J
```

キーインしましたらRUNしてください。

最初に年月日を聞いてきますから、3つ入力しましょう。すると、次のエラーメッセージが表示されます。

```
NEXT without FOR in 120
Ok
█
```

これは、120行のNEXTに対応するFOR文がないということです。リストをよく見てください。NEXTのうしろのループ変数がJになっています。これがIでないとうまくFOR～NEXTが成立しません。キーインミスということになりますね。

FOR～NEXT文の中でさらにFOR～NEXT文を使ったりするようになりますと、このエラーによく出会います。要は、FOR文の数とNEXT文の数が等しいか、またループ変数は正しく使われているかをチェックすることです。

120行をなおしておきましょう。

```
120 NEXT I
```

Type mismatch [?TM Error]

もう一度RUNしてみましょう。
画面には次のメッセージが表示されました。

```
? 58,10,20
ｽｳﾘｮｳ,ｷﾝｶﾞｸ? 5000,100
Type mismatch in 110
Ok
■
```

タイプ・ミスマッチ、これはタイピング、つまりキーインし間違えたということではありませんよ。タイプが合わない、肌が合わないということです。

どう肌が合わないかといいますと、行番号110をごらんください。左辺は文字型配列、右辺は数値変数の積ですからやはり数値です。つまり、文字型の変数に数値を代入していたのです。右辺と左辺の変数の型（タイプ）が合わなかったのです。

ドルマークをとっておきましょう。

```
110 KINGAKU(I)=SU*TA
```

Overflow [?OV Error]

さあ、今度はパッチリいくでしょう。RUN!!
うーん、またまたエラーです。

```
? 58,10,20
ｽｳﾘｮｳ,ｷﾝｶﾞｸ? 5000,100
Overflow in 110
Ok
■
```

これは前に勉強しましたから、なぜだかわかりますね？

このプログラムでは、行番号20で、すべての数値変数の整数型宣言をしていました。ですから、配列KINGAKUも当然、整数型配列になります。ところが行番号110の右辺の結果は整数型の上限32767を超えてしまっています。これを左辺の整数型配列に押し込もうというのですから、無理が生じるのは当然ですね。

行番110、それから、配列KINGAKUを使いますので、エラーメッ

セージがでないうちに配列宣言をしておきましょう。

```
15 DIM SHOHIN$(20),KINGAKU#(20)
110 KINGAKU#(I)=SU*TA
```

Undefined label [?UN Error]

今度は、行番号110で計算した金額をサブルーチンへ飛ばして画面に表示させてみましょう。

次のステートメントをキーインします。

```
115 GOSUB *DAT.DISPLAY
```

サブルーチンでは、商品名と金額を、タイマーを入れて少しのあいだ表示しておくようにします。

次のステートメントを打ち込んでください。

```
500 '
510 *DATA.DISPLAY
520 CLS
530 PRINT SHOHIN$(I)
540 PRINT "  ==>";KINGAKU#(I);"円"
550 FOR J=1 TO 2000 : NEXT J
560 CLS
570 RETURN
```

いいですか？

それでは実行してみましょう。今度は、次のエラーメッセージです。ウンザリしてきましたね。

```
? 59,01,05
ｽｳﾘｮｳ,ｷﾝｶﾞｸ? 5000,10
undefined label in 115
Ok
█
```

GOSUB文による飛び先に「定義されていないラベルを使った」というわけです。コンピュータにしかられても文句はいえませんよ。決してこちらに勝ち目はないのですから……。まあ、ここのところは折

れてバグを探したほうがよさそうです。

おッ、ありました、ありました。ラベル名が、ジャンプしていくと思っていた行番号510のラベル名と違っていました。

どちらに統一してもいいのですが、サブルーチン側のラベルに合わせましょうか。行番110を次のように修正します。

```
115 GOSUB *DATA.DISPLAY
              ↑
              └──────────挿入します
```

RETURN without GOSUB (?RG Error)

そしてRUN!! 今度は11回くりかえしてください。

どうにか終ったと思ったら次のエラーメッセージです。

```
RETURN without GOSUB in 570
Ok
█
```

11回データをインプットして、一応目的は達成したのですからいいのですが、それでもエラーがでるとちょっと気分が悪いですね。

これは、90〜120行のFOR〜NEXTループを終えて、処理がその下の行へ落ちていき、つまり最初に出会う500行からを一度だけ実行し、570行のRETURN文でひっかかったのです。

RETURN文までを実行しても、これはGOSUB文でジャンプしてきての実行ではないですね。そこでGOSUB文がない、ということにあいなったわけでございます。

ですから、上のメッセージが出る前に、0が画面に表示されたと思います。これはKINGAKU#(12)の中味です。なぜ添字が12かといいますと、FOR〜NEXTを抜けたときループ変数には最終値に1だけ加算したものが格納されるからなのでございます。ハイ。

このエラーを防ぐには、次のEND文を入れてしめくくります。

```
130 '
140 END
```

「ケジメがなさすぎる!!」と、コンピュータ様々にしかられてしまったわけです。「今後、気をつけます、ハイ」

第8章

サウンドとタートルグラフィック

8・1 拡張命令について

拡張命令とは、従来のＰＣ－8801のコマンドにさらに追加された命令群のことで、タートルグラフィック機能、サウンド機能など、かなりおもしろいものがありますので、本章でこれらの使い方について簡単に説明しておくことにします。

サウンド機能

これまでＰＣ－8801ではゲームなどの効果音を出そうとしてもＢＥＥＰ音しか使うことができませんでしたが、マークIIに追加されたこの機能を利用すると、ゲームその他のプログラムに、いろいろ飾りつけをすることができます。

たとえばゲームをプレイする前に、その内容に合わせた短い曲を演奏するプログラムを実行するようにしておけば、突然ゲームが開始される場合とくらべて、ずいぶん雰囲気がもり上がります。

なかには音楽が苦手だという人もいるかもしれませんが、楽譜さえあれば、五線譜に書かれた音程を対応コードになおしてキーインするだけですから、とても簡単です。

タートルグラフィック機能

これは画面に表示されたタートル（亀）を操作して、いろいろな方向へ進ませながらグラフィック図形を描くものです。

方向指定も、「前へ50進む」「うしろへ100進む」といった意味のステートメントが用意されていますから、簡単にグラフィックスを楽しむことができます。

そのほか、高速画面消去のコマンドも加わりましたので、これも使ってみることにしましょう。

本章では、拡張命令を使って、簡単なサブルーチンを作り、私たちの作った「ブロックくずし」に追加して、少しばかり飾りつけをしてみたいと思います。

8・2　拡張命令の追加手順

拡張命令を使えるようにするためには、システムディスクまたは標準添付のカセットテープにセーブされている次のユーティリティプログラムをＲＵＮさせなければなりません。

❶ @load.n88　（システムディスク）
❷ cas2:@exst（カセットテープ）

これはいずれも、マシン語プログラムをメモリの＆ＨＤＣ００から＆ＨＥ５ＦＦ番地までにロードするためのプログラムです。この操作をしてはじめて、拡張命令を使うことができるようになります。

それでは、ロード方法について説明します。

ディスク使用の場合

次のようにキーインしてください。

```
files ↵
```

ドライブ1のディスクのファイル名を表示させるのでしたね。

画面は次のようになりました（下の画面は、横40桁のモードでＦＩＬＥＳコマンドを実行してあります）。

```
files
dskut2.n88 9     @load .n88 1
@exst *    2     block .    1
SANSUU.    1     DMAIL .    1
block1.    1
Ok
■
```

この中に

@load.n88

というファイル名がありますね。これが拡張機能を追加するためのユーティリティプログラムです。

さあ、それではこれをロードしてＲＵＮさせますが、この２つの操作を１回でやってしまう方法があります。次のようにキーインしてく

ださい。

```
run "@load.n88"
```

RUNのうしろにファイル名（プログラム名）をつけてリータンキーを押すと、このプログラムをロードして、さらに実行してくれます。

実行結果は次のような画面になります。

```
run "@load.n88
The following statements are available.

    1. CMD TURTLE
    2. CMD SING
    3. CMD CLS
    4. CMD TEXT ON/OFF
    5. CMD CUT

Please enjoy new features.
Ok
```

- 1. CMD TURTLE ← タートルグラフィック命令
- 2. CMD SING ← サウンド命令
- 3. CMD CLS ← 高速画面消去命令
- 4. CMD TEXT ON/OFF ← テキスト画面の切り換え命令
- 5. CMD CUT ← 拡張機能をカットする命令

●上の画面は「ユーティリティプログラムを実行したので５つの命令が使えるようになりました。どうぞお楽しみください」といっているのです。

これで準備完了です。

カセットテープの場合

標準添付のカセットテープから、次のようにしてユーティリティプログラムをロードします。

```
load "cas2:@exst"
```

「Ok」が表示されてロードが終了しましたら、 f･5 キーを押してプログラムをRUNさせてください。

実行結果の画面はディスクの場合と同じで、上に示した画面のようになります。

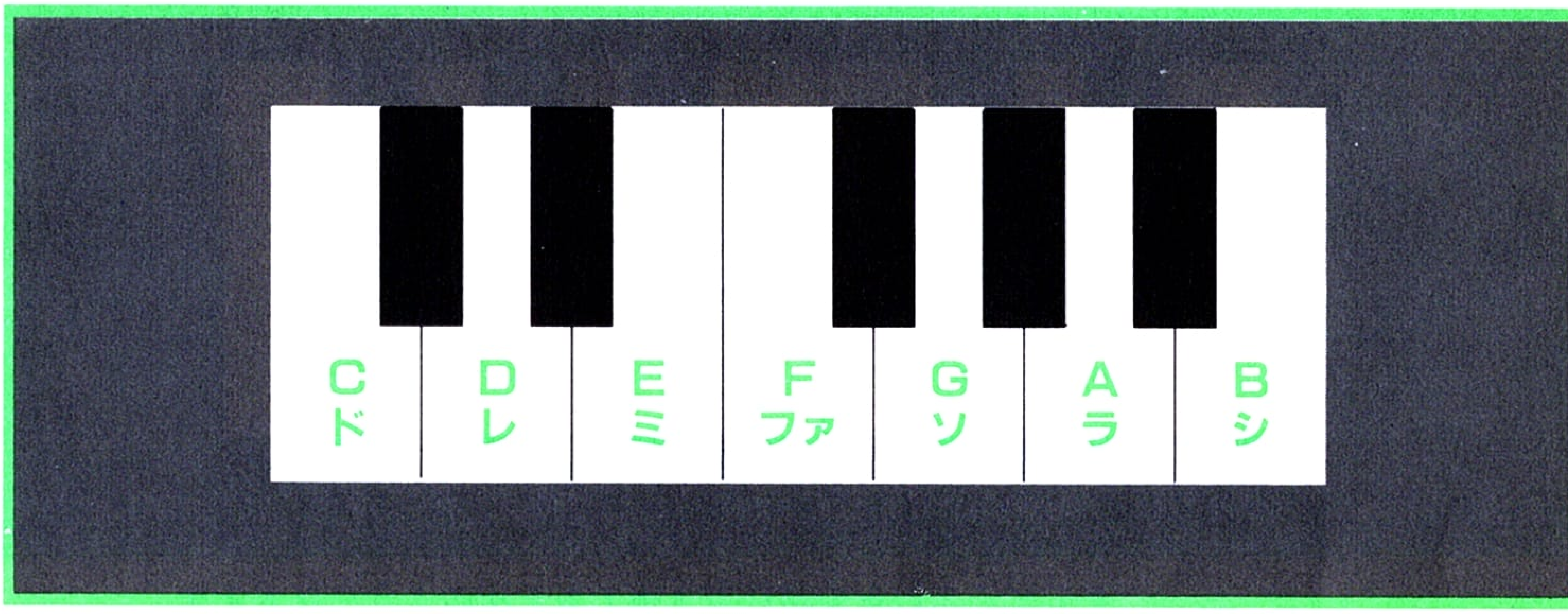
これで準備完了です。

この拡張命令使用のためのマシン語プログラムは、電源を切ってしまうと消えてしまいますが、それ以外は、いつでも拡張命令を使用することができます。
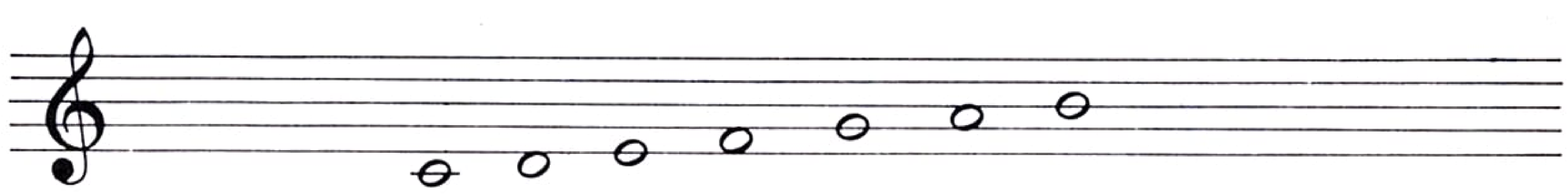
電源を切ってしまった場合は、もう一度システムを起動し、いま説明したのと同じ操作を行なってください。

8・3　サウンド機能を活かそう

次の図は、みなさんよくごぞんじの鍵盤ですね。黒いところは半音、白いところが全音ですね。



これを楽譜になおすと



となりますね。

私は、小中学生の頃音楽がきらいでいつも先生におこられていたくらいですから、どうもこういうのはニガテなのですが、サウンド機能の説明ですから、がまんしてがんばりましょう。

上の図を見てもわかるように、「ドレミファソラシ」をコンピュータでは「CDEFGAB」に割り当てています。ですから、これと前ページの画面の「2　CMD　SING」を結びつけることによって、音階をだすことができるのです。

CMD　SING
（シーエムディ・シング）

さっそくテストしてみましょう。

```
cmd sing "cdefgab" ⏎
```

ダブルクォーテーション内の文字列は、大文字でも小文字でもかまいません。

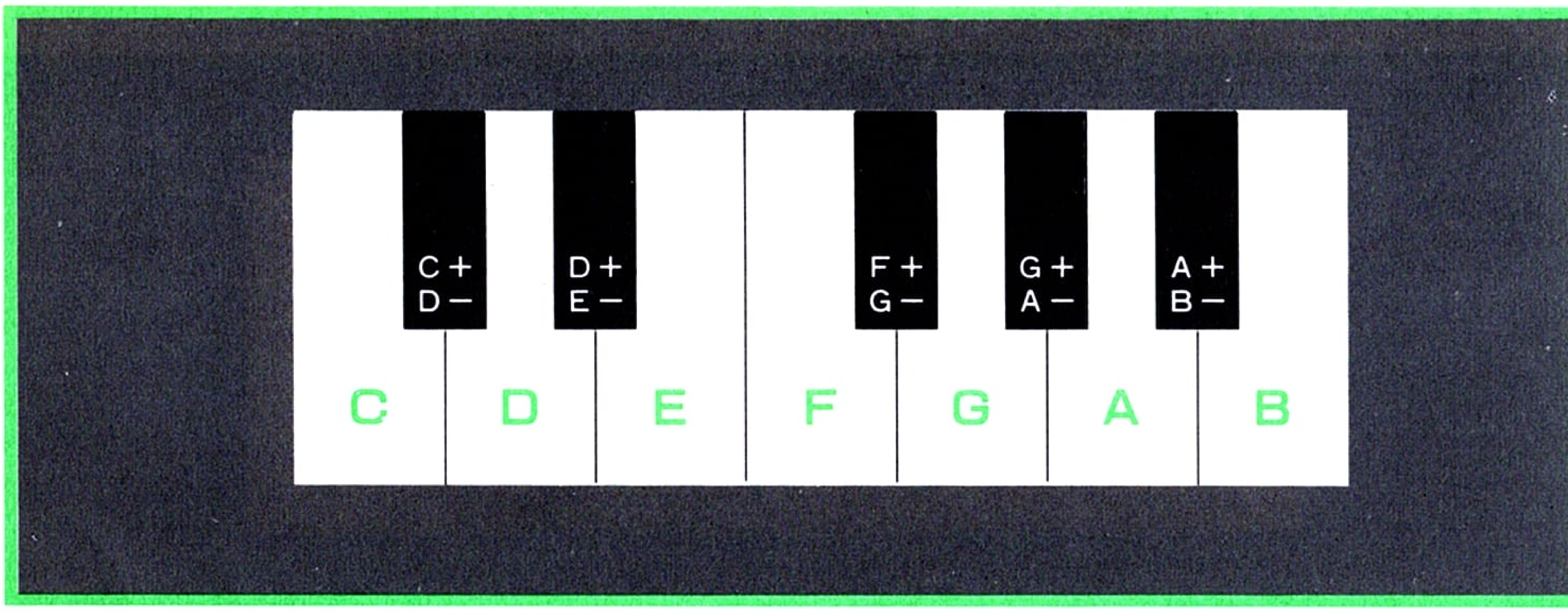
どうですか？　リターンキーを押すやいなや、「ドレミファソラシ」と演奏されたでしょう。演奏というほどのものでもありませんが、とにかく音がでましたね。

〈注〉この節では「CMD SING」をよく使いますのでファンクションキー2にこれを定義しておくと便利です。方法は141ページをごらんください。

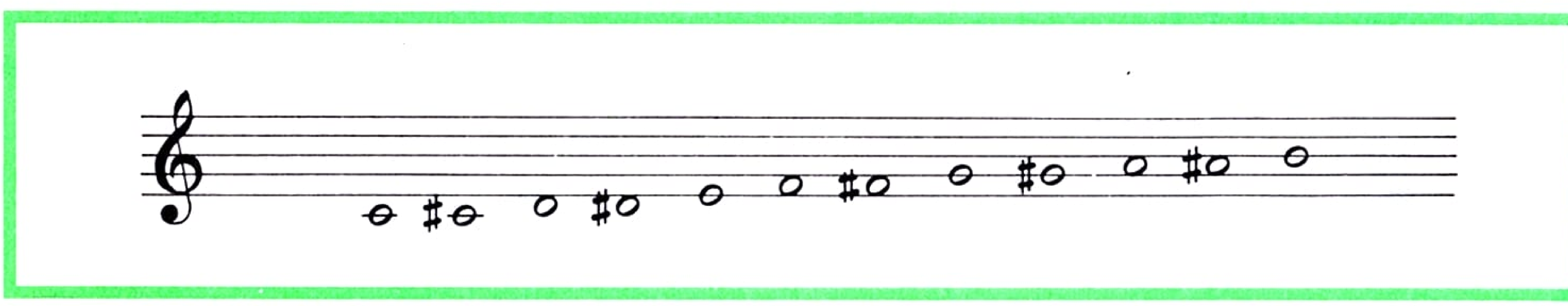
これでまずまず安心です。ちょっと耳ざわりの音ですが、これはあ

とで説明しましょう。

ところで、半音を表現するにはどうすればよいでしょうか？
次の図をごらんください。



そして、これを楽譜に書き換えると次のようになります。



もうわかりましたね。
そうです、半音上げるには音階を表わす記号のうしろにプラスを、半音下げるにはマイナスをつければよいのです。
たとえば「C」の半音高い音は「C＋」で、これは「D－」と同じことですね。
それでは、上の楽譜をプログラムにしてみましょう。

```
cmd sing "cc+dd+eff+gg+aa+b"
```

どうですか？　うまくいきましたか？
これがサウンド命令の原理です。けっこう簡単だと思いませんか？
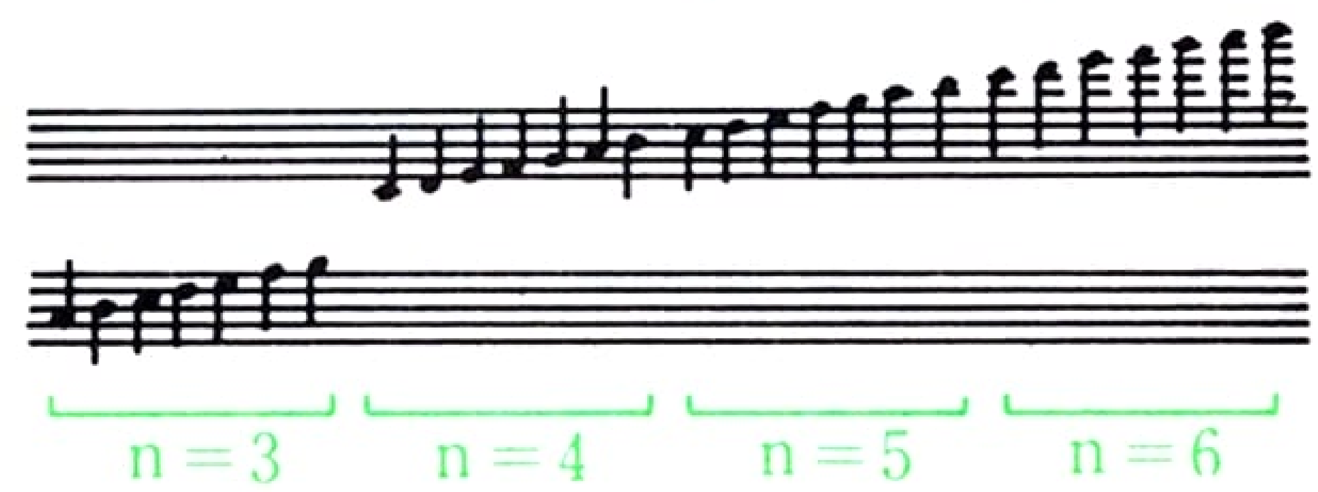
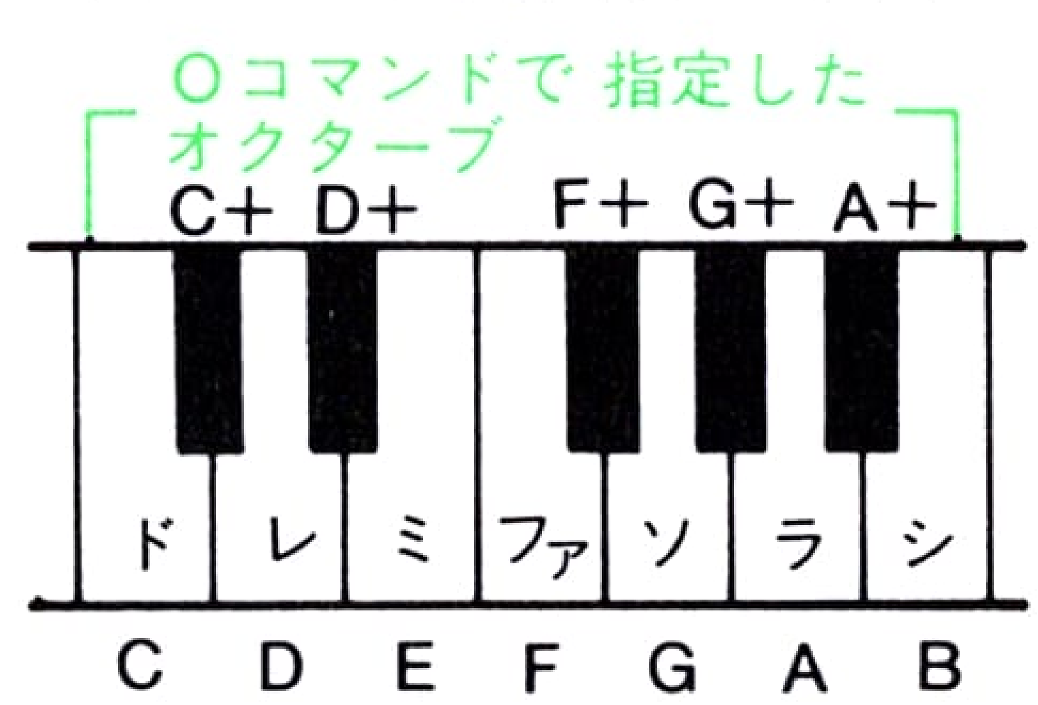
オクターブを設定する

右ページの表の上から2つ目の欄をごらんください。マークIIで使用できる音域が示されています。
n＝3から6までの4オクターブを使うことができるのですね。

サブコマンドO

音域を指定するには「サブコマンドO」を使います。「O」のうしろに3～6の数字をつけて指定します。

ミュージックサブコマンド

コマンド	条　件	機　　　能
Tn	48≦n≦255	テンポを設定します。数が大きいほど速くなります。　nの値で，1分間に演奏する4分音符の数を指定します。
On	3≦n≦6	オクターブを設定します．nの値とC～Bで演奏するオクターブの対応は，次のとおりです． 
Ln	1≦n≦32	音符，休符の長さのデフォルト値を設定します．nの値で，何分音符（休符）かを指定します． n＝1　n＝2　n＝4　n＝8　n＝16　n＝32 
R［n］［.］	1≦n≦32	休符．nの値で，休符の長さを指定します．指定の方法は，Lコマンドと同じです．省略した場合は，Lコマンドで指定した値とします．符点休符には，“.”（ピリオド）を付けます．
C～B ［±］［n］［.］	1≦n≦32	音符．C～Bで音符を指定します．“＋”で半音上がり，“－”で半音下がります．  nの値で，音符の長さを指定します．符点音符には，“.”（ピリオド）を付けます．
Xn	n＝0または n＝1	テキスト画面の表示を制御します．X0で，テキスト画面を消し，X1でもとに戻します．澄んだ音を出すときは，最初にX0を行います．
RPj［…］	1≦j≦255	［…］をj回くりかえします． ネスティングは，8レベルまでです．

＊　コマンドは，小文字でもかまいません．

拡張機能のユーティリティプログラムを実行したときは、音域の初期値（デフォルト値）は「O４」に設定されています。

次のステートメントを実行してみてください。

```
cmd sing "o3co4cdeo5co4c"
```

このステートメントでは、「D」「E」にはオクターブ指定がありませんから、一番新しく指定された「O４」が有効になります。

音の長さを指定する

これまで私たちは音の長さを指定しませんでしたから、どの音符も同じ長さで演奏されました。今度は、音の長さを変える方法を学びましょう。

サブコマンドL
〈注〉「CMD SING」がメインコマンドですから、それに付随したコマンドという意味でサブコマンドという表現が使われています。

この方法は２つありますが、まず最初は「サブコマンドL」です。

これもオクターブ指定と同じように「L」のうしろに数字をつけて何分音符か指定します。この数値には１以上32以下の整数値が使われ、たとえば「２」でしたら２分音符が、「16」でしたら16分音符が対応しています。

音の長さのデフォルト値は４、つまり４分音符に設定されています。

では、次の２つのステートメントをテストしてみてください。

```
cmd sing "cdefgab"
```

```
cmd sing "l1cdefgab"
```

２つ目はずいぶん間のびした感じに演奏されましたね。すべて全音符として実行されたのです。

このように、音の長さを最初に指定すると、そのあとに続く音符は変更がないかぎり、同じ指定で演奏されます。

音の長さを指定するには、もう１つ方法があります。それは、音を表わす記号のうしろに数値をつけるのです。たとえば、「C」の音を２分音符にしたければ「C２」、8分音符にしたければ「C８」というようにです。

次のステートメントをテストしてみてください。

はじめの「ドレミ」は４分音符で演奏されましたが、「ファソラシ」は全音符になってしまいましたね。

```
cmd sing "c4d4e4fgab"
```

これはなぜかと申しますと、さきほど指定した「L１」がまだ生き

きているからです。

ですから、「ファソラシ」を16分音符にしたければ、次のようにサブコマンドLを挿入すればよいのですね。

こうしますと今度は、音の長さの初期値は16分音符に設定されますから、以後変更がないかぎり、この長さで演奏されます。

```
cmd sing "c4d4e4l16fgab"
```

どうですか？　感じがつかめましたね。

休符を入れる

次のステートメントを実行してみてください。

```
cmd sing "cccccc"
```

「ド」の音が全音符と同じような長さで聞こえましたね。上のCの音はさきほどの「L16」が生きていますから、16分音符の「ド」の音が6つ続けて実行された形になりました。

私たちは「ド、ド、ド、ド、ド、ド」と断続的に演奏されるのを期待したのですが、そうはいきませんでした。

この問題を解決してくれるのが「サブコマンドR」です。　サブコマンドR

これもやはり、音符の長さの指定と同じで、うしろに数字をつけて使います。

いまのプログラムを次のように変更して実行してみてください。

```
cmd sing "cr16cr16cr16cr16cr16c"
```

いかがですか？　付符が入ってだいぶ歯切れがよくなったでしょう。

すきとおった音にする

これでほぼ大切なコマンドの説明は終りましたが、みなさん、いままで実行したステートメントの音質が少し悪いと思いませんでしたか。

次の2つのステートメントを実行して比較してみてください。

```
cmd sing "cdefgab"
```

```
cmd sing "x0cdefgab"
```

2つ目のステートメントはずいぶん澄んだ音になりましたね。そのかわり、演奏中は、いままで画面に表示されていた文字が消えてしま

いました。

サブコマンドX

このように「サブコマンドX」のパラメータを0に指定することにより、画面への表示を停止し澄んだ音を出すことができます。

なお、画面への表示を解除するときはパラメータに「1」を指定します。解除の指定がなければ、プログラムが終るまで画面には何も表示されません。

いまのサンプルプログラムは短いですから、ステートメントの実行が終了し次第、画面表示は復帰しますが、長いプログラムなどでは、「X1」の指定をしないと画面表示のステートメントがあっても、何も現れませんから注意してください。

リピートコマンド

サブコマンドRP

同じ節（ふし）を何回かくり返したいときは「サブコマンドRP」を使います。

次のステートメントを実行してみてください。

```
cmd sing "rp[x0cdefgab]"
```

「RP」のうしろにパラメータを入れることにより、くり返しの回数を指定することができます。

スクリーンエディタで、いまのプログラムに「3」を挿入してためしてみましょう。

```
cmd sing "rp3[x0cdefgab]
```

どうですか？　かなり便利ですね。

しかし、どの音も16分音符で演奏されます。これは前ページの一番上のステートメントで「L16」を指定しましたので、音の長さが16部音符に初期設定されたままだからです。電源投入時の状態になおしておきましょう。

```
cmd sing "l4"
```

符点音符

音楽では、音符のとなりに符点をつけて音の長さを調節しています。コンピュータでももちろん、この処理はフォローされています。

たとえば4分音符のド（C）に符点がついている場合には、「C4.」というようにピリオド（.）をつけます。

次の例をテストしてみてください。

```
cmd sing "x0c.d.e.fgab"
```

テンポを調節する

テンポとは、曲が演奏されるスピードのことです。テンポの初期値は、1分間に4分音符を120回演奏する速さに設定されていますが、「サブコマンドT」により、これを変更することができます。

サブコマンドT

次の2つのステートメントをためしてください。

```
cmd sing "t200x0cdefgab"
```

```
cmd sing "t80x0cdefgab"
```

違いがはっきりわかりますね。

このサブコマンドのパラメータは、48から255までの値を指定でき、数字が大きくなるほど速いテンポになります。

簡単なプログラム

私たちはいままで、CMD　SINGの使い方についてダイレクトステートメントでいろいろテストをくり返してきましたが、このコマンドも、扱いはまったく普通のプログラムと同じです。

CMD　SING以下の文字列を変数に代入することもできますし、くり返し演奏をFOR～NEXT文で行なうこともできます。

次のプログラムをキーインしてテストしてみてください。

```
100 CMD SING "t150"          ←テンポの設定
110 READ A$,B$               ←サブコマンドデータを変数に代入
120 FOR I=1 TO 3
130   CMD SING A$            3回演奏する
140   CMD SING B$
150 NEXT I
160 END
170 DATA x0cdefgabo5cr4      ストリングデータ
180 DATA x0co4bagfedcr4
```

8・4 「ブロックくずし」に装飾音を

さて、前節でサウンド機能の使い方をご理解いただけたと思いますので、ここでは、これを使って私たちが第4章で作った「ブロックくずし」のプログラムに装飾音をつけ加えてみましょう。

ところで、拡張機能を使用するためのユーティリティプログラムはロードして実行してありますね？

今日の学習をこの節から始めた方はまだだと思いますから、8・2節を見ながらユーティリティプログラムを実行してください。こうしておかないとサウンド機能は使えません。

準備できましたら、第4章の「ブロックくずし」のプログラムをロードします。

カセットの場合

カセットテープの方は

```
load "cas1:block" ⏎
```

ディスクの場合

ディスクバージョンの方は

```
load "block" ⏎
```

でよいですね？。

音楽演奏サブルーチン

ここに、音楽演奏といえるほどではないのですが、簡単なイントロを用意しましたので、これをサブルーチンにして追加することにしましょう。次のようにキーインしてください。ダブルクォーテーションの中のアルファベットは大文字でも小文字でもいいですよ。

```
3000 '
3010 *MUSIC
3020 CMD SING "x0t200o418a4aga4ao5co4agaaa2"
3030 CMD SING "ao5co4ao5co4ao5cde2"
3040 RETURN
```

サブコマンドXで画面表示を消して、テンポは200に指定してあります。また音域は「O４」「O５」を使っていますね。

みなさんのお気にめすリズムかどうか、とにかくオリジナルの作曲ですよ。

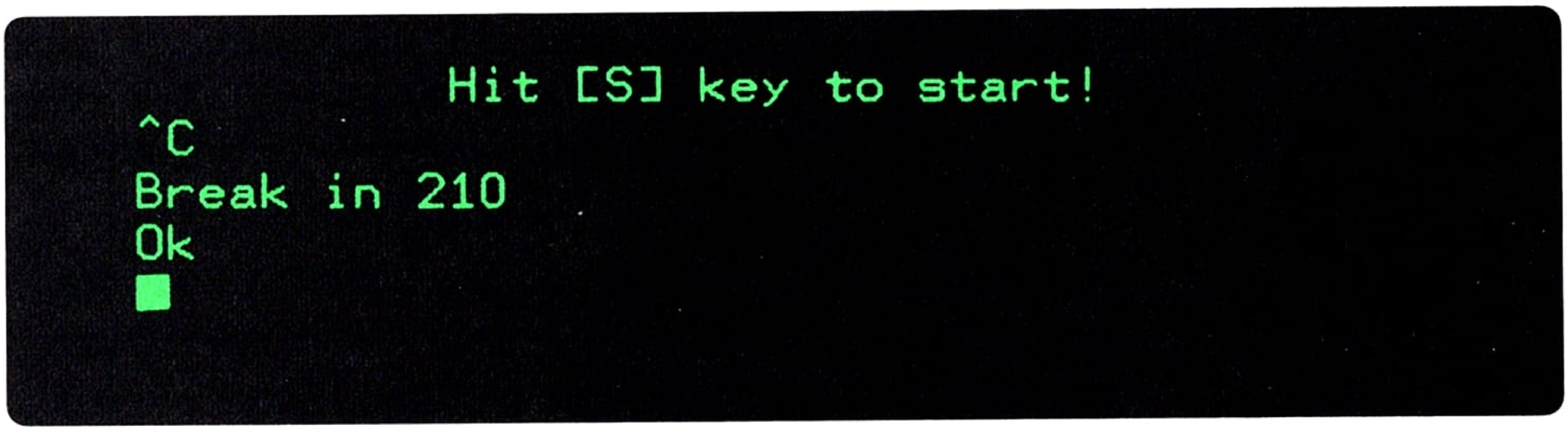
ミュージックのイントロをサブルーチンにしましたので、次のステートメントをキーインしておきましょう。

```
185 GOSUB *MUSIC
```

さあ、これでＲＵＮさせてみてください。奇妙な音楽が演奏されたでしょう。

演奏が終ってゲームが始まるかと思うと、待てどくらせどいっこうにそのけはいはありません。それどころか画面には何も表示されないのです。

しかたないですから STOP キーを押してプログラムの実行を停止させてみましょう。画面は次のようになりました。



ははあ、プログラムはキー入力待ちのルーチンで止まっていたのですね。それにしても画面に「Ｓ」キーを押せとの指示メッセージがでませんでしたが、どうしてでしょうか？

これは、3020行のステートメントで「Ｘ０」、つまり画面表示を消すサブコマンドを実行していますので、サブルーチンから戻っても、この状態が維持されたままだったからです。

CMD TEXT ON

これを解消するのに「ＣＭＤ　ＴＥＸＴ　ＯＮ」があります。これはテキスト画面をＯＮにする、つまり表示させるということですね。ＯＦＦにすれば当然、画面表示は消えることになります。

次の１行を追加しましょう。

```
186 CMD TEXT ON
```

今度はうまくいくと思います。実行してみてください。

なお、この「ＣＭＤ　ＴＥＸＴ　ＯＮ」は拡張命令パッケージのマシン語がロードされていないと使えませんから注意してください。

それではプログラム名を変えてセーブしておきましょう。

```
save "cas1:block1"
```
← カセットへセーブする

```
save "block1"
```
← ディスクへセーブする

8·5　タートルグラフィックの使い方

さて、次はタートルグラフィック機能の使い方ですが、これは『ユーザーズマニュアル』にかなり親切に説明してありますので、本書では、サブコマンドの一覧表を右ページにかかげ、使い方を簡単に説明し、あとは「ブロックくずし」のプログラムにこの機能を利用したサブルーチンを追加し、このゲームを、よりいっそう充実したものにすることにします。プログラムは消さないでください。

タートルグラフィック命令の書式は次のとおりです。

```
cmd turtle "サブコマンド"
```

画面に表示されたタートル（亀）を、ダブルクォーテーションの中にいろいろなサブコマンドを入れて動かし、グラフィック図形を描いていくものです。

私たちがプログラムリストなどを表示させる画面のことを「テキスト画面」といいますが、このタートルグラフィックで描かれる絵は、「グラフィック画面」に表示されますので、画面の後方にグラフィック図形があり、その上にテキスト画面の文字が少し浮き上がった形で表示されるように感じられます。

画面消去の方法は、次のように３つあります。

❶　テキスト画面だけを消去するとき

```
cmd cls 1
```

❷　グラフィック画面だけを消去するとき

```
cmd cls 2
```

❸　テキスト、グラフィック画面両方を消去するとき

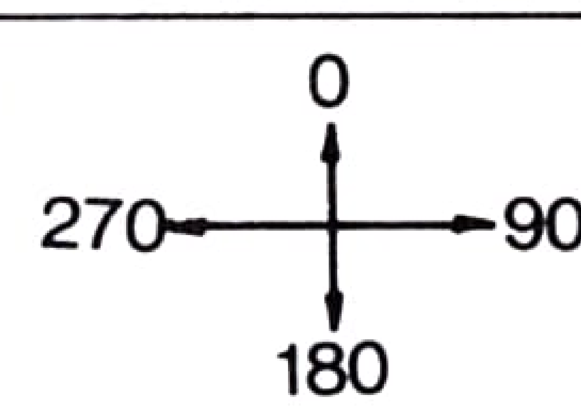
```
cmd cls 3
```

なお、これらの画面消去コマンドは、拡張パッケージを使うユーティリティプログラムが実行されてないと使うことができません。

私たちはいままで「ＣＬＳ」を使ってきましたが、上の３つのコマンドは、これにくらべてかなり高速に処理されます。

さて、右ページの表をごらんください。いろいろなサブコマンドがありますが、これらはいずれも欧文の頭文字をとって表現されていますので、使い方は簡単ですから、これさえ覚えてしまえばあとは楽だ

グラフィックサブコマンド

コマンド	条　件	機　　　能
FDn	−32768≦n ≦32767	タートルをn進めます．負の数を指定すると後退します．タートルの向きは変化しません．
BKn	−32768≦n ≦32767	タートルをn後退させます．負の数を指定すると前進します．タートルの向きは変化しません．
MVx,y	−32768≦x, y ≦32767	タートルをスクリーン座標（x，y）へ移動させます．タートルの向きは変化しません．
SXx	−32768≦x ≦32767	タートルをスクリーン座標（x,（現在のy座標））へ移動させます．（水平移動）タートルの向きは変化しません．
SYy	−32768≦y ≦32767	タールをスクリーン座標（（現在のx座標），y）へ移動させます．（垂直移動）タートルの向きは変化しません．
HDn	−32768≦n ≦32767	タートルの向きを決めます．正の数で時計まわり．負の数で反時計まわり． 
LTn	−32768≦n ≦32767	タートルの向きを現在向いている方向からn度左に向けます．
RTn	−32768≦n ≦32767	タートルの向きを，現在向いている方向からn度右に向けます．
CPn	n＝0または n＝1	640×200モードで，x方向の補正をします．FD，BKコマンドで有効です．通常は，CP1（補正する）の状態で使います．
PD	なし	タートルのペンを下げます．このあとPUコマンドが実行されるまで，タートルは軌跡を残しながら移動します．
PU	なし	タートルのペンを上げます．タートルは軌跡を残さずに移動します．
PCn	0≦n≦7	タートルのペンの色を指定します．nは，パレット番号です．
HT	なし	タートルを消します．
ST	なし	タートルを表示します．
RPj[…]	1≦j≦255	[…]をj回くり返します．ネスティングは8レベルまでです．

※　コマンドは，小文字でもかまいません．

と思います。

1 FD

これは、「FORWARD」の略で、タートルを前進させます。

2 BK

「BACK」の略ですね。タートルを後退させます。

3 MV

「MOVE」の略で、タートルを指定した座標位置に移動させます。

4 SX

「SCREEN X」の略で、タートルを水平移動させます。

5 SY

これは「SCREEN Y」で垂直移動をさせるのですね。

6 HD

これは私の想像ですが「HEAD DIRETION」ではないかと思います。つまりタートルの頭の向きを変えるのです。

7 LT

これは「LEFT TURN」でしょう。左方向へ一定角度回転させます。

8 RT

「RIGHT TURN」ですね。右へ向きを変えます。

9 CP

「COMPENSATION」で、X方向の補正をします。

10 PD

「PEN DOWN」。これはタートルをペンにたとえています。つまりペンを紙の上に下ろした状態です。

11 PU

「PEN UP」。ペンを上げた状態ですね。

12 PC

「PEN COLOR」の略で、描く色を決めます。カラーコードは39ページの表を参照ください。

13 HT

「HIDE TURTLE」で、タートルを消してしまうのですね。

14 ST

「SET TURTLE」。これはタートルを表示します。

15 RP

「REPEAT」の略で、図形を描く動作を指定回数だけくり返します。

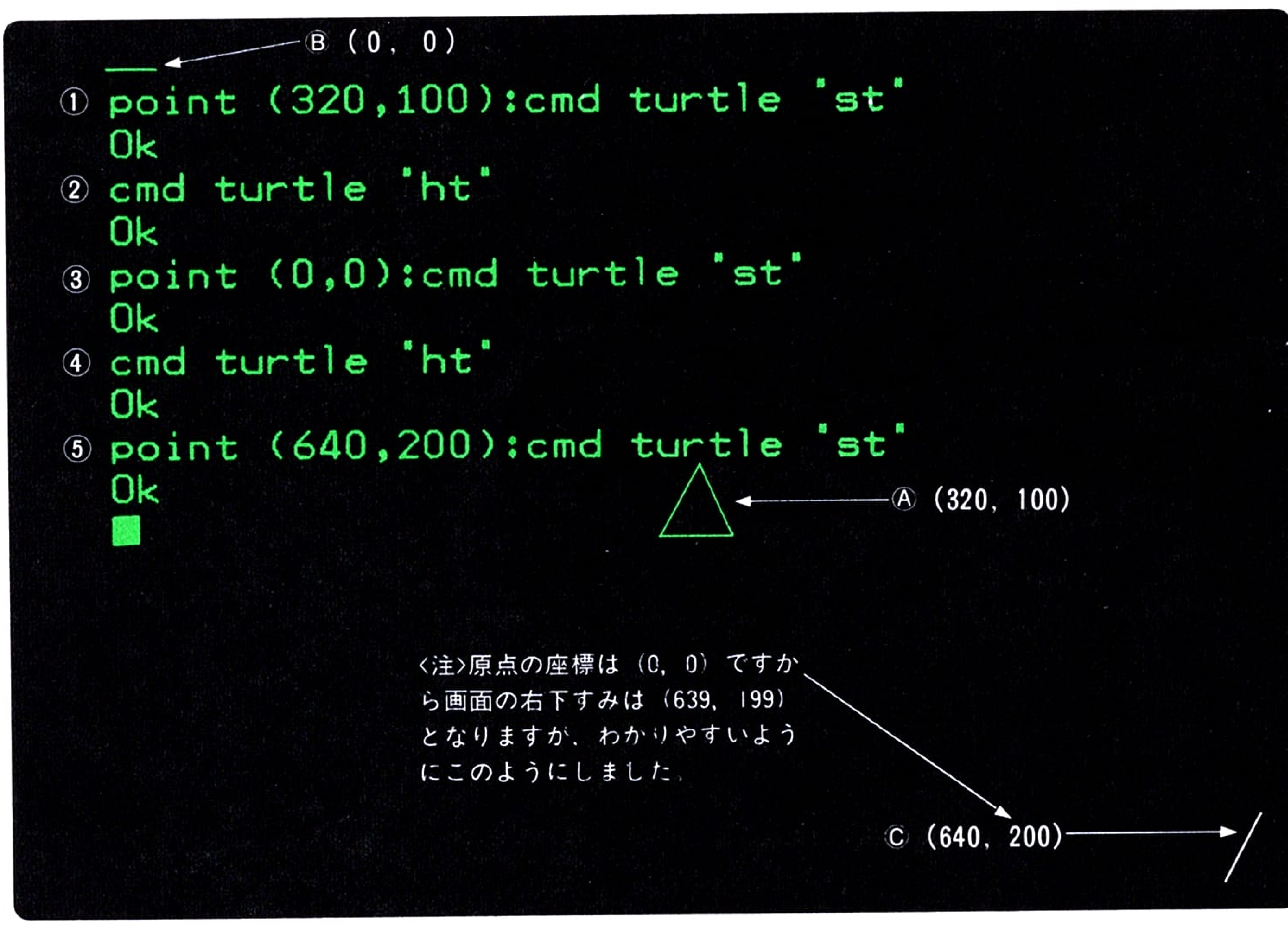
どうですか？　これでサブコマンドがずいぶんわかりやすくなったと思います。

ところで、このタートルグラフィックでの座標系はどのようになっているかをダイレクトステートメントで確認してみますが、ＣＭＤ ＴＵＲＴＬＥはよく使いますので、ファンクションキー２に定義しておきましょう。次のようにキーインします。

```
key 2,"cmd turtle "+chr$(34)
```

スペースを１つ入れる
ダブルクォーテーションを表示させます

これで準備ができました。次の画面を見ながら❶〜❺のステートメントを実行してみてください。



❶のステートメントは、スクリーン座標（320、100）の位置をＰＯＩＮＴ文で指定し、そこにタートルを表示させるものです。

私たちはいままで、最大で横80桁×縦25行のキャラクタ座標を使っていましたが、このタートルグラフィックでは、もっと目の細かいスクリーン座標を使い、これは１つ１つがキャラクタではなくドットと呼ばれる小さな点で表現されます。

そして、タートルグラフィックでは横640ドット、縦200ドットのきめ細かさになっています。

❶のステートメントでは、ちょうど画面の中央を指定していますか

ＰＯＩＮＴ

●ポイント文はグラフィック座標系での位置を指定するものです。グラフィックではこのほかＬＰ（ラストリファレンスト・ポイント）という概念を使いますが、本書では説明を省略します。

ら、Ⓐの位置にタートルが表示されます。

そして、他の位置にタートルを表示させたいときは、❷のように一度タートルを消去してから、❸のようにＰＯＩＮＴ文で座標を指定しサブコマンドＳＴを使って表示させます。

❸を実行すると、ディスプレイの左上すみにタートルの尾のほうが表示されました。

同じように、❹❺を実行すると、今度は右下すみにタートルが表示されます。

これで、だいたいタートルグラフィックの座標系の感じがつかめたと思いますので、次のステートメントをダイレクトモードでキーインしてから、『ユーザーズマニュアル』の15－4ページからの記事を実行してみてください。

```
cmd turtle "ht" ↵
```

いかがでしたか？　案外簡単だったでしょう。

それでは私たちの最終目的「ブロックくずしプログラムにグラフィックサブルーチンを追加すること」を実行したいのですが、紙数の関係で解説することができません。プログラム総リストを示し、追加変更点は緑色にしておきますので、キーインしてみてください。

```
100 ' ﾌﾞﾛｯｸ ｸｽﾞｼ
110 ' file name ==> block2
120 ' 1984.1.20
130 ' save "block2"
140 '
141 CLEAR ,&HDBFF        ┐
142 BLOAD "@exst",R      ┘── これはディスク使用の方だけキーインしてください
150 WIDTH 40,25:CONSOLE 0,25,0,1
160 COLOR 6
170 DEFINT A-Z:VRAM=&HF3C8
180 '
181 GOSUB *MUSIC        ┐
182 GOSUB *TURTLE       ┘── 音楽演奏・作画サブルーチンへ
185 GOSUB *MUSIC
186 SCORE=0:CMD TEXT ON:COLOR 6
190 LOCATE 10,5:PRINT "Hit [S] key to st
art!"
200 Y$=INKEY$
210 R=RND(1)
220 IF Y$<>"S" THEN GOTO 200
230 '
```

```
240 CMD CLS ←―――――――――――――――――――― 高速画面消去にしました
250 PRINT " ◢";STRING$(22,"■");
260 PRINT "◣ [ SCORE    0]"
270 FOR I=1 TO 23
280   LOCATE 1,I:PRINT "■"
290 NEXT I
300 FOR I=1 TO 23
310   LOCATE 24,I:PRINT "■"
320 NEXT I
330 LOCATE 1,24:PRINT "◤";SPC(22);"◥";
340 COLOR 3
350 FOR I=1 TO 6
360   LOCATE 2,AGAIN+2+I
370   PRINT STRING$(22,"□")
380 NEXT I
390 BALL.Y=23:MOVE.Y=-1:LIMIT=10
400 BALL.X=RND(1)*22+2
410 MOVE.X=INT(RND(1)*2)*2-1
420 '
430 *MAIN
440 I=INP(0)
450 COLOR 5:BEEP 0
460 RACKET.X=(I=247)*(LIMIT<21)-(I=253)*
(LIMIT>2)
470 LOCATE LIMIT,24:PRINT "   ";
480 LIMIT=LIMIT+RACKET.X
490 LOCATE LIMIT,24:PRINT "■=■";
500 COLOR 7
510 GOSUB *CHECK
520 IF P=32 THEN GOTO *SET.BALL
530 IF P>227 AND P<232 THEN MOVE.Y=-MOVE
.Y:MOVE.X=-MOVE.X
540 IF P=134 OR P=61 THEN MOVE.Y=-MOVE.Y
:GOTO 510
550 IF P=135 THEN MOVE.X=-MOVE.X:GOTO 510
560 IF P=219 THEN BEEP 1:GOSUB *BLOCK
570 IF SCORE MOD 66 =0 THEN AGAIN=AGAIN+
2:GOTO 240 ELSE GOTO 510
580 '
590 *SET.BALL
600 LOCATE BALL.X,BALL.Y:PRINT P$;:BALL.
X=BALL.X+MOVE.X:BALL.Y=BALL.Y+MOVE.Y
610 LOCATE BALL.X,BALL.Y:PRINT "●";
```

```
620 IF P>60 THEN P=32
630 IF BALL.Y<24 THEN P$=CHR$(P):GOTO *MAIN
640 '
650 LOCATE 7,15:PRINT "************"
660 LOCATE 7,16:PRINT "  GAME OVER!"
670 LOCATE 7,17:PRINT "************"
675 FOR I=1 TO 1000:NEXT I
680 '
690 CMD CLS 1
695 Y$=INKEY$:IF Y$<>"" THEN GOTO 695
700 LOCATE 7,5
710 PRINT "Do you want to play again?"
720 LOCATE 15,7:PRINT "< y/n >"
730 Y$=INKEY$:IF Y$="" THEN GOTO 730
740 IF Y$="N" OR Y$="n" THEN GOTO 770
750 LOCATE 7,5:PRINT SPC(30)
760 LOCATE 15,7:PRINT SPC(10):GOTO 185
770 CMD CLS 1
780 LOCATE 12,5:PRINT "Have a nice day!!"
790 FOR I=1 TO 3000:NEXT I
800 END
1000 '
1010 *CHECK
1020 P=PEEK((BALL.Y+MOVE.Y)*120+(BALL.X+
MOVE.X)*2+VRAM)
1030 RETURN
2000 '
2010 *BLOCK
2020 LOCATE INT((BALL.X+MOVE.X)/2)*2,BAL
L.Y+MOVE.Y
2030 PRINT "  ";:SCORE=SCORE+1
2040 LOCATE 33,0
2050 PRINT USING "###0";SCORE
2060 MOVE.Y=-MOVE.Y:BEEP 0
2070 RETURN
3000 '
3010 *MUSIC
3020 CMD SING "x0t200o418a4aga4ao5co4aga
aa2"
3030 CMD SING "ao5co4ao5co4ao5cde2"
3040 RETURN
4000 '
4010 *TURTLE
```

●695行のステートメントを入れておきませんと、ゲームが終ってから画面に、プレイ中に押したキーの数字「1」「3」がたくさん表示されてしまいます

先行入力されたストリングを掃除する（695行）

710行のメッセージを消す（750行）

720行のメッセージを消す（760行）

```
4020 POINT (160,100)            ←スタートラインの設定
4030 CMD TURTLE "st"            ←タートル表示
4040 CMD TURTLE "rp90[rt4pc1fd5]"  ┐
4050 CMD TURTLE "rp72[rt5pc2fd5]"  │
4060 CMD TURTLE "rp60[rt6pc3fd5]"  ├─6色の円を描く
4070 CMD TURTLE "rp51[rt7pc4fd5]"  │
4080 CMD TURTLE "rp45[rt8pc5fd5]"  │
4090 CMD TURTLE "rp40[rt9pc6fd5]"  ┘
4100 CMD TURTLE "ht"            ←タートルを消す
4110 '
4120 CMD TEXT ON                ←これを入れないとテキスト画面が表示されません
4130 A$="ﾌﾞﾛｯｸ ｸｽﾞｼ"
4140 L=LEN(A$)                  ←文字列の長さを変数に代入
4150 FOR J=1 TO 3
4160   LOCATE 16+J,5+2*J
4170   FOR I=1 TO L
4180     PRINT MID$(A$,I,1);    ←文字を1文字ずつとり出して表示
4190     FOR K=1 TO 100 :NEXT K ←タイマー
4200   NEXT I
4210 NEXT J
4220 '
4230 CLS 2                      ←6つの円をゆっくりと消す
4240 '                            CMD CLSではムードがでません
4250 FOR J=1 TO 3                         ┐
4260   LOCATE 16+J,5+2*J                  │
4270   FOR I=1 TO L                       ├─「ブロッククズシ」の文字を
4280     PRINT " ";                       │   1文字ずつ消す
4290     FOR K=1 TO 100 :NEXT K           │
4300   NEXT I                             ┘
4310 NEXT J
4320 RETURN
```

忘れずにプログラムを新しい名前でセーブしてください。

ディスクバージョンの方が後日このプログラムを実行するには

```
run "block2"
```

でよいのですが、テープの方は次のようにしてください。

```
load "cas2:@exst"
run
load "cas2:block2"
run
```

●テープバージョンの方は新しいカセットを用意し、まず「cas2:@exst」をセーブし、続いて「cas2:block2」としてこのプログラムをセーブしておくと便利です。

PC-8801MKⅡプログラミング入門

編著者　ナツメ社編集部　© Natsumesha 1984
発行者　田村正隆

発行所　株式会社　ナツメ社
東京都千代田区神田神保町1－52(〒101)
電話　03(291)1257（代表）
振替　東京3－58661
印　刷　ラン印刷社
製　本　三修紙工

ISBN4-8163-0360-X　Printed in Japan

PC-8801mkII
プログラミング入門

発行――1984年2月20日
編著者――ナツメ社編集部
発行者――田村正隆
発行所――株式会社ナツメ社
郵便番号101
東京都千代田区神田神保町1-52
電話〈03〉291-1257
振替 東京3-58661
〈落丁・乱丁本はお取り替えします〉

定価――1400円

PC-8801mkII
プログラミング入門

［目次より］

ナツメ社
定価＝1400円

ISBN4-8163-0360-X C2054 ¥1400E